NASA Contractor Report 159365

Theoretical Study of Network Design Methodologies for the Aerial Relay System

Jorge M. Rivera
and
Robert W. Simpson

Flight Transportation Laboratory
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

CONTRACT NAS 1-15268
JUNE 1980

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

NASA Contractor Report 159365

THEORETICAL STUDY OF NETWORK DESIGN METHODOLOGIES FOR THE AERIAL RELAY SYSTEM

Jorge M. Rivera
and
Robert W. Simpson

FTL Report R80-10

June 1980

Flight Transportation Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

This Page Intentionally Left Blank

INTRODUCTION

Growth of the United States air transportation system is currently facing two major barriers: energy and congestion. While the price of fuel has gone up by approximately an order of magnitude in the last 10 years, there is no assurance that fuel will continue to be available at the levels desired by the airlines. At the same time, lack of capacity at the major airports is causing delays to increase, both in number and duration. Both of these factors are causing the price of air transportation to reverse a 40-year-old trend and to increase in real terms, negating gains in aircraft productivity and engine efficiency.

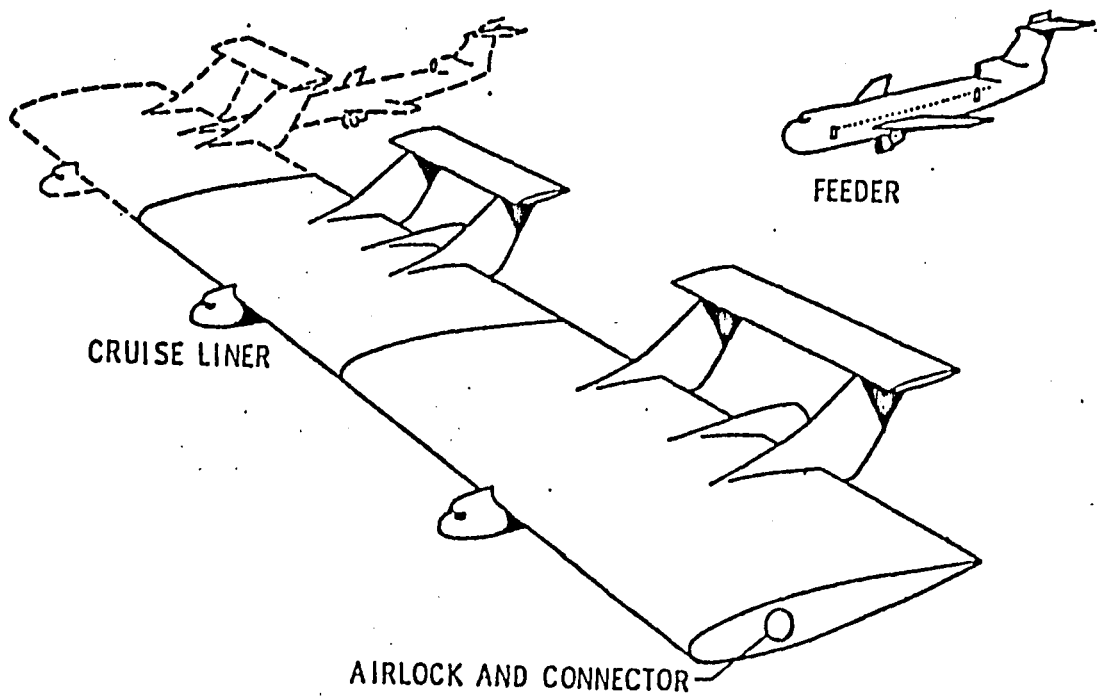
These considerations have led some observers of the aviation scene to conclude that the air travel mode is reaching maturity, although various regulatory, economic, and technological options have been suggested which offer incremental improvements to the existing system. For substantial growth to continue, however, major structural changes may be necessary. One imaginative and radical departure is the Aerial Relay System (Albert C. Kyser, "The Aerial Relay System: An Energy Efficient Solution to the Airport Congestion Problem," NASA Technical Memorandum 80208, January 1980).

Briefly, in the Aerial Relay System a series of "liners", made up of "line modules", continuously cruise over the

United States at a set altitude and on a predetermined schedule. These liners are met by a fleet of "feeders" carrying aloft passengers bound for cities along the liners' routes and accepting passengers destined for their own base. The basic elements of the system are shown in Figure I. A fully-developed Relay system could provide frequent non-stop service between practically any two cities in the United States.

The advantages of the Relay system are many. The elements of the system can be tailored for their own function leading to efficiency of operation: the liners for cruise conditions, the feeders optimized for short-haul takeoff and climb. But the basic attraction lies in the Relay system's ability to unload the major hubs' airports by utilizing secondary (or satellite) airports and smaller city airports for the feeder's operations; since one of the major functions of airports, especially those at large hubs, is the interchange of connecting passengers between airplanes, this transfer is now performed onboard the liners. The feeder from a smaller city or secondary airport takes up passengers bound for many destinations downstream (and accepts diverse passengers for the downward journey), bypassing the hub and relieving the hub of these operations. The Relay system would thus supplement and not replace the existing airline networks; the hub-to-hub origin-destination traffic could continue to be served by dedicated aircraft at the major airports. Alternatively, the Relay system could serve as the major link between

Figure I. Aerial Relay System Elements: Liner and Feeder Aircraft



large hubs while utilizing satellite airports and thus relieving the major airports of this type of traffic.

Thus the Aerial Relay System has intrinsic appeal, as it could both relieve congestion and decrease energy consumption of the air mode. Clearly, substantial engineering and design work is required before the system can be implemented. However, some questions regarding the fundamental mathematical network properties of the Relay system can be addressed to insure that no basic drawbacks to the general concept exist. This report presents the derivation of a generalized algorithm which can be used for basic design studies of networks for the Aerial Relay System.

TABLE OF CONTENTS

Table of Contents, continued

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY:

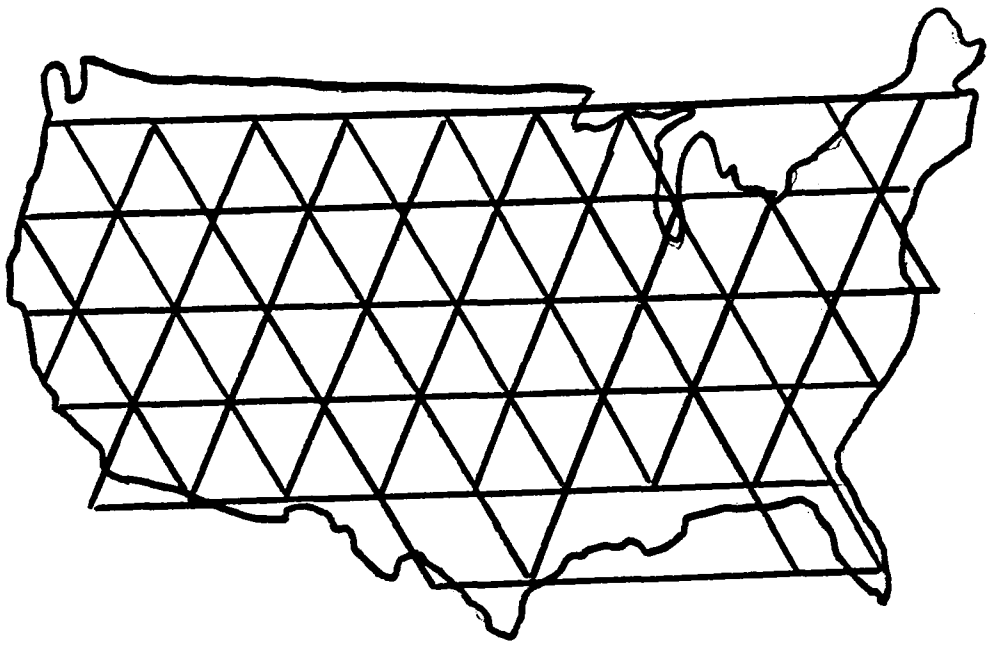
THE RELAY SYSTEM NETWORK DESIGN PROBLEM

The essential ingredients of an analytical approach to network design and analysis include the basic concepts of graph theory and flows in networks. Both are concepts which have received considerable attention in the literature. By the same token, the network design problem also has attracted the interest of a large number of researchers since its solution was thought to be relevant to the design of urban transportation networks.

"Network Analysis" is the study of a given set of nodes and arcs that connect the nodes. The arcs represent the Relay system liner routes, and the nodes represent terminals or exchange points where the passengers/cargo connect from one liner to another. The origins of network analysis are old and diverse. Network analysts rely heavily on graph theory, a branch of mathematics that was founded by Euler in 1736. In the 1940's operations research yielded a number of techniques, such as linear programming, for the mathematical study of network systems. Concepts of this kind, together with probability theory, statistics and computer programming, are the tools of network analysis. The factors which need to be considered in network analysis include the performance of the network, in terms of its economics, and the structural properties of the network, in terms of its vulnerability to disruption.

"Network design", on the other hand, is concerned with obtaining a good layout for the route network. In its simplest form, the classical optimal network design problem consists of building a connected "subnetwork" from a given large-scale hypothesized network. The subnetwork is developed by selecting a subset of links in the large-scale network that minimizes the sum of the shortest routes between all node pairs. A cost or "budget" constraint limits the number of links that may be included. The objective reflects the costs of using the network and the budget constraint limits the construction costs of the network. For example, it is conceivable that one may begin with a network grid for the Relay system sketched below, a regular, triangular pattern which completely covers the continental USA, irrespective of demand volumes and locations. Speeds are chosen so that vehicles can meet regularly at network nodes. A subnetwork of connected nodes can be built from the large-scale hypothetical network by applying the concepts of graph theory and flows in a network.

This approach is the classical approach; others also exist. One of the simplest of these methods is called a "branch exchange." Here, we are given a set of nodes and their locations, and it is important to connect them by arcs in order to achieve a specified design goal. The method consists of modifying an initial network design using these nodes by repetitively removing branches and replacing them with the same number of new branches in new locations. The problem is difficult because of the many configurations that can be used to connect the nodes, but



computer programs are available which are capable of making a number of surprising improvements.

Another approach is a heuristic design procedure which is similar to the philosophy of mathematical programming. The first step is to find some solution, regardless of cost, by any convenient means. Then, changes to this starting solution are considered and those which lower the cost are retained. We begin by choosing a subset of the service points, known as "key cities", that are spread across the entire country. A series of heuristic operations is then performed which improves the routing for circuits whose paths include a number of these key cities. After this has been accomplished, sections of the structure are sequentially removed and treated in the same manner. This is called "subproblem processing", a subproblem being similar to the entire problem but having fewer nodes and a set of requirements involving only these nodes. The subproblem establishes a desirable skeleton network on which to build the remainder of

the solution. The solution is modified by adding branches to provide alternate paths. In the resulting network all requirements are implemented along the path of shortest total length. Thus, this technique is unlike the "branch removal" technique in that a solution is obtained by adding more branches to a subproblem, finding the cheapest path, and routing the requirements (liners) along it.

We postulate a network pattern covering the United States, which will be flown by Relay liners. A daily schedule for liner flights exists such that meetings between liners occur at network intersections or nodes allowing passenger loads to be interchanged. There is a daily pattern for the volume of traffic flow between city pairs. At any time, an individual passenger can determine the best path from his origin to a desired destination consisting of one or more connected liner flight segments, and can make reservations to ensure available space.

The problem is "Construct an economically-efficient network pattern and schedule of service", given:

1) Liner speeds:

2) Demand patterns between city pairs.

The answers should consist of both a network pattern (or route map) and a timetable (or schedule map) for liner flights. The paths followed by individual liners, and the interchange of modules at intersections are also determined. A network flow of passengers, and a schedule of feeder flights, is also found.

Thus, the Relay network development problem is classified as a "Network Design" problem. It turns out to be an extremely

difficult problem. An extensive review of existing methodologies in network design indicated that the "state-of-the-art" in existing mathematical techniques and computational capability cannot solve such a network design problem due to both complexity in its formation and the scale of the networks contemplated.

The research reported here represents a step towards developing the required mathematical techniques. It extends an existing technique of the "branching" or "searching" category called "Implicit Enumeration" to the case where multiple optimization criteria can be specified. This was chosen as a point of departure for research since any mathematical technique for designing a Relay network will have to consider a variety of optimization criteria, such as minimum passenger travel time and minimum liner and feeder operating costs. An efficient computational algorithm has been developed and tested, but due to computational costs it is only successful on networks which are substantially smaller than the Aerial Relay network contemplated. Further research must be directed towards decomposition techniques where a sequence of smaller networks are considered.

SECTION 1

BACKGROUND OF MULTICRITERION PROGRAMMING

1.1 The Network Design Problem

It is clear nowadays that we can not solve all the problems in terms of a single objective function. The complexity of daily life and the necessity of finding more real solutions will necessarily oblige us to solve problems with several objective functions at the same time. Thus, several solutions will appear and the best solution of the problem will not be known with certainty and will depend upon the preferences of the decision maker.

There are many areas in which we can apply this class of problems. Multiobjective problems can be found in areas such as Financial Decision Making, Educational Planning, Transportation Systems, Production and Inventory Planning, etc... Of all of these, Transportation is one of the most important areas in which a multicriterion framework can be

applied. In Air Transportation for example, there is a large number of criteria to be considered: decrease travel time, decrease total costs or simply fuel costs, improve safety and comfort, enhance air quality and reduce noise impact, etc...

An area in which a multicriterion framework had been already suggested is Transportation Networks Design. Agarwal [1] , presents the application of two interactive optimization techniques to the design of transportation networks under multiple objectives. He makes an excellent discussion of the importance of considering more than just one single criterion, which most commonly has been the minimization of costs.

He suggests a representative list of criteria which is reproduced below and sets up an example of a transportation network design.

1. Decrease Travel Time.
2. Decrease Travel Cost.
3. Improve Other Service Characteristics.
4. Improve Safety.
5. Increase Accessibility.

6. Provide comparable transportation services to all segments of the population in relation to their needs.

7. Provide transportation facilities and services to encourage development in accordance with comprehensive plans and to control development to support transportation plans.

8. Enhance air and water quality.

9. Minimize expenditures of public money for the construction and operation of transportation systems.

10. Minimize consumption of energy/fuel supplies.

11. Minimize noise impact.

12. Enhance property values.

13. Decrease personal tax burdens.

14. Minimize dislocation and permanent disruption of neighborhoods.

15. Minimize disruption due to construction activities.

16. Improve quality of neighborhoods.

In essence, the problem is stated as the one of selecting the best combination of development projects which produces the best traffic network, evaluated in terms of given criteria while maintaining expenditure within the given budget.

The criteria suggested by Agarwal in his problem are:

-Minimizing Total Travel Time.

-Minimizing Total Construction Costs.

-Minimizing Total Vehicle Miles.

-Minimizing Total Number of Dwelling Units Taken For Rights-of-way Over The Entire System.

The problem formulated is then solved using Geoffrion et. al. [23] and Benayoun et. al.[8] interactive approaches. Agarwal concludes that both procedures are promising in the application to the design of transportation networks and suggests the extension of his results in some other areas.

1.2 The Vector Maximum Approach

An important and basic element for any approach to the multicriterion programming problem is the concept of a non-dominated solution. Before studying this concept, we will first expose the following concepts provided by Yu and Zeleny [46].

Let $G(x)=(f_1(x),\ldots,f_p(x))^t$ denote a functional set of

p criteria defined over the decision space S, $S \subset R^n$. Given a point in the criteria space, say $g \in G \subset R^p$, we could associate with it a set of domination factors, DF(g), such that if $g' \neq g$, g' and $g \in G$, and $g' \in g \oplus DF(g)$, then g' is dominated by g. The symbol $\oplus$ means that the addition operation is done over all the elements of the set. An example of a DF is a p-dimensional vector d, with the property that if $g' = g + ud$ where $u > 0$, $d \in DF(g)$, then g dominates g'. It is clear that any positive multiple of d is also a DF.

A nondominated solution is one that is not dominated by any other feasible choice. For instance, g' is a non-dominated point if for all $g \in G$, $u > 0$, and $d \in DF(g)$

$$g' \neq g + ud$$

Among the different approaches to the multicriteria programming problem is the vector maximum approach. This approach tries to find or generate all the efficient solutions.

EFFICIENT POINTS: A point $x^o \in S$ is said to be efficient if there is no other point $x \in S$ such that $G(x) \geq G(x^o)$, with

at least one strict inequality. We can see that x^o is a non-dominated solution, with respect to $DF(G(x))$.

$$DF(G(x)) = \{ d \in R^p,\ d \leq 0 \}$$

The vector maximum approach tries to generate all the points $x^o \in S$, such that there does not exist another $x \in S$, satisfying

$$G(x^o) = G(x) + ud \qquad u > 0$$

and $d_i \leq 0, \quad i = 1,\ldots,p$

It is important to notice that the vectors d, can be regarded as feasible directions of dominance, because if

$$G(x) + ud = G(x')$$

then, since $ud \leq 0$, $G(x) \geq G(x')$.

THE 0,1 MULTICRITERIA PROBLEM

Multiple objective programs with continuous variables

have been exhaustively treated in the literature. However, very little has been done for the zero-one case.

The linear multiple objective problem with zero-one variables is written as

$$(P) \qquad \text{Max} \quad \{ Cx : x \in F \}$$

where $F = \{ x \in R^n : Ax \leq b,\ x_j = 0,1 \quad j \in J \}$, C is a pxn matrix, A is a mxn matrix, b is a mx1 vector and J = 1,...,n

The solution set to (P) is the set of efficient points denoted by EF(P). Specificically $x^o \in F$ is said to be efficient in P, if there is no $x \in F$ such that $Cx \geq Cx^o$ with at least one strict inequality. From this point on, the partial ordering relation $x \geq y$, will mean $x_j \geq y_j$, $j \in J$ with at least a strict inequality.

A typical practical application that can be reduced to this model is the "Project Selection Problem". The columns a^j of A corresponds to projects to be selected or rejected by p interested parties on the basis of the px1 evaluation

vectors c^j (the columns of C).

A central difference between convex and discrete multiple objective programs is that in the former case, if the Kuhn-Tucker constraint qualifications hold, every efficient point i (P) maximizes a linear functional of the type λCx, for a $\lambda \in R^p$, $\lambda > 0$, on the feasible set. In the later case, this may not happen as is shown in the example below:

$$\text{Max} \left\{ \begin{pmatrix} 1 & -2 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} : x_1 = 0,1 \text{ and } x_2 = 0,1 \right\}$$

The point $(x_1, x_2) = (0,0)$ is efficient in the last problem but does not maximize a functional λCx, for any $\lambda \geq 0$, on F. This type of points is what Bitran [9] defines as weak efficient points. Conversely, a strong efficient point is one that solves the problem $\max \{\lambda Cx\}$ $x \in F$, for a given set of weights λ.

As we have seen, not every element of EF(P) maximizes a functional of the type λCx, with $\lambda \geq 0$, on F. However for any $\lambda > 0$, every solution to

$$(P_\lambda) \qquad \text{Max } \{ \lambda Cx : x \in F \}$$

is efficient in (P). The linear functional λCx corresponds to the assignment of weights to the p criteria. As these weights are ussually subjective, it would be useful to investigate the set of weights for which a given element of EF(P) solves (P_λ).

Similarly to what is done in Kornbluth's paper [32] on the continuous multiple objective linear program, we characterize, for the zero-one case, the indifference set

$$S = \left\{ \lambda \in R^p : \sum_{i=1}^{p} \lambda_i = 1; \quad \lambda_i \geq 0, \ i = 1,\ldots,p \right\}$$

where $x^n \in EF(P)$ and $S(x^n) = \{ \lambda \in S : x^n \text{ solves } (P_\lambda) \}$

1.3 Algorithms for Linear MultipleObjective Programs

1.3.1 Zero-One Variables

An algorithm for finding the efficient solutions for problem (P) was developed by Pasternak and Passy [39]. They studied problem (P) for the case of two criteria and presented

an algorithm based on a parametric approach combined with an extension of Balas filter method. They applied the algorithm to solve the project selection problem and reported the solution in [13].

Shapiro [41], presents basic theoretical results that provide a means of generating efficient solutions to the problem, based upon integer programming duality theory. However, he does not provide a procedure that generates the entire efficient set of integer solutions.

Bitran [9,10], has developed basic theoretical results as well as general procedures for obtaining the efficient set of solutions to zero-one multicriteria programming problems. Both methods are based on the study of the auxiliary problem

$$(P') \qquad \text{Max} \left\{ Cx : x_j = 0,1 \quad j \in J \right\}$$

and in the generation of auxiliary vectors that are, afterwards, mapped into the set of non-efficient and efficient solutions of the problem.

The auxiliary problem (P') is clearly related with the original problem (P), since every efficient point in (P') that is feasible originally, is also efficient in (P).

The general scheme is based on the generation of vectors $v \in R^n$ such that $v = (v_1,\ldots,v_n)^t$, $v_j = 0,1$ or -1 and $cv \geq 0$. These vectors are considered as directions of preferences. For instance, if $x = x' + v$, then, since $cx = cx' + cv$; $cv \geq 0$, we have that $cx \geq cx'$, or x dominates x'. Obviously,

$$cx = cx' + (-cv)$$

In both methodologies, Bitran relates the vectors of preferences with the set of efficient and non-efficient solutions for the auxiliary problem (P'). In the first procedure [9], he employs the following mapping for such purposes:

$$v_j = 0 \longrightarrow x_j = 0 \text{ and } 1$$
$$v_j = 1 \longrightarrow x_j = 0$$
$$v_j = -1 \longrightarrow x_j = 1$$

where v and $x \in R^n$ space.

This map relates the set of vectors, v, with the set of

dominated points of the auxiliary problem (P').

As a by-product of the results, an algorithm to determine EF(P) was obtained, however, its applicability was limited to small problems.

In the second approach [10], Bitran uses another mapping:

$$v_j = 0 \longrightarrow x_j = 0 \text{ and } 1$$
$$v_j = 1 \longrightarrow x_j = 1$$
$$v_j = -1 \longrightarrow x_j = 0$$

In this case, the set of vectors $v \subset R^n$ space, is related to the set of efficient points of (P'). In both schemes, the generation of the vectors, v, is done via implicit enumeration.

Once the set of zero-one efficient solutions for problem (P') is determined, Bitran proceeds to relate this set with that of the original problem by checking feasibility. The efficient points of the auxiliary problem that are feasible in (P), are efficient points of (P). Finally, the set of

efficient solutions of the original problem is completed through the identification of those non-efficient points for (P') which are part of EF(P).

The algorithm is presented in terms of a directed graph having as modes the vertices of the unit hypercube, which are either isolated nodes or end nodes of d-paths, and as arcs a subset $\overline{V}$ of V, the set

$$V = \left\{ v \in R^n : v_j = 0,1 \text{ or } -1 \quad j \in J, \; cv \geq 0 \right\}$$

The approach that uses the first mapping is denoted as the forward algorithm. The other is called the backwards algorithm.

Computational evidence reported by Bitran shows the superiority of the backwards algorithm. A detailed analysis of the computational time spent in the different parts of this algorithm reveals that the backward parts are those which consume the most time and which limit the algorithm to a relatively small number of variables.

1.3.2 A Dynamic Programming Approach

A new approach to solve multicriterion integer linear programming problems is the one presented by Villarreal and Karwan [44].

The algorithm is essentially an extension of the fundamental dynamic programming recursive equations. In the second part of their paper, relaxations and fathoming criteria which are fundamental to branch and bound procedures are suggested to obtain an alternate hybrid approach.

The problem is formulated as follows:

$$(P) \qquad \text{v-max} \quad \sum_{n=1}^{N} c^n x_n$$

$$\text{subject to} \quad \sum_{n=1}^{N} a^n x_n \leq b$$

$$K_n \geq x_n \geq 0 \quad \text{INTEGER}$$

A transformation in the set of constraints is done in order to make the problem suitable for a dynamic programming

approach.

In the first part of the paper, three different recursions are presented.

The first recursion enables them to find the set $n(Yn)$ of efficient solutions for the n-stage problems with a vector of resourses Yn from the set Hn-1 (Yn-1) of efficient solutions for the (n-1) stage problem with a vector of resources Yn-1, and Ho (Yo)= $\{0\}$ for all Yo. The set of constraints have been transformed to:

$$Y_{n-1} \leq Y_n - a^n x_n$$

$$K_n \geq x_n \geq 0 \quad \text{INTEGER}$$

Using this recursion they can obtain the sets of solutions for any right hand side vector $\{0\} \leq Yn \leq b$ for any n-stage problem. They have to be applied for n=1,...,N and each vector of resources Yn.

An alternative recursive formation is suggested. The

recursion starts at Y= $\{0\}$ and then, proceeds increasing the values of the vector, Y, until Y=b. This procedure can be used to obtain the set of efficient solutions for each vector of values, $\{0\} \leq Y \leq \{b\}$, at stage n=N. Thus, a basic difference between this recursion and the first one is that it is not necessary to go through each stage n($\leq$ N), in order to obtain the set of solutions for any vector, Y, at stage N.

These two recursive formulations require the explicit determination of the vector of resources Y. This implies that each vector, Y, must be identified with its corresponding set of efficient solutions, and hence as the number of constraints increases, the necessary storage requirements are increased enormously. Another recursion is suggested to alleviate this problem and consequently they develop a procedure that can be used to obtain the set of efficient solutions for any vector of resources, $\{0\} \leq Yn \leq b$, such that there will not be any requirements to explicitly define any of the vectors of resources until the last stage (N).

The procedure considers first all the possible feasible values of Xn at stage n, n=1,...,N, producing n-dimensional

points by combining feasible values of Xn with the (n-1)-dimensional efficient points. Then, after eliminating all of those n-dimensional points which are infeasible, the set of efficient solutions of the problem at stage-n, is obtained via pairwise comparisons.

In the second part of the paper, a dynamic programming hybrid approach is presented, by introducing bounding procedures.

An efficient solution for the n-stage problem, with resource consumption vector Yn is said to be fathomed by bounding when its functional set value Φ an upper bound of the residual problem is less or equal than any member of the set of lower bounds.

Different techniques for finding upper and lower bounds are reviewed and some of them are coded in a new algorithm.

An important issue that they take into account is the trade off to be made between the benefit that more bounds provide, eliminating possible solutions, and the price that

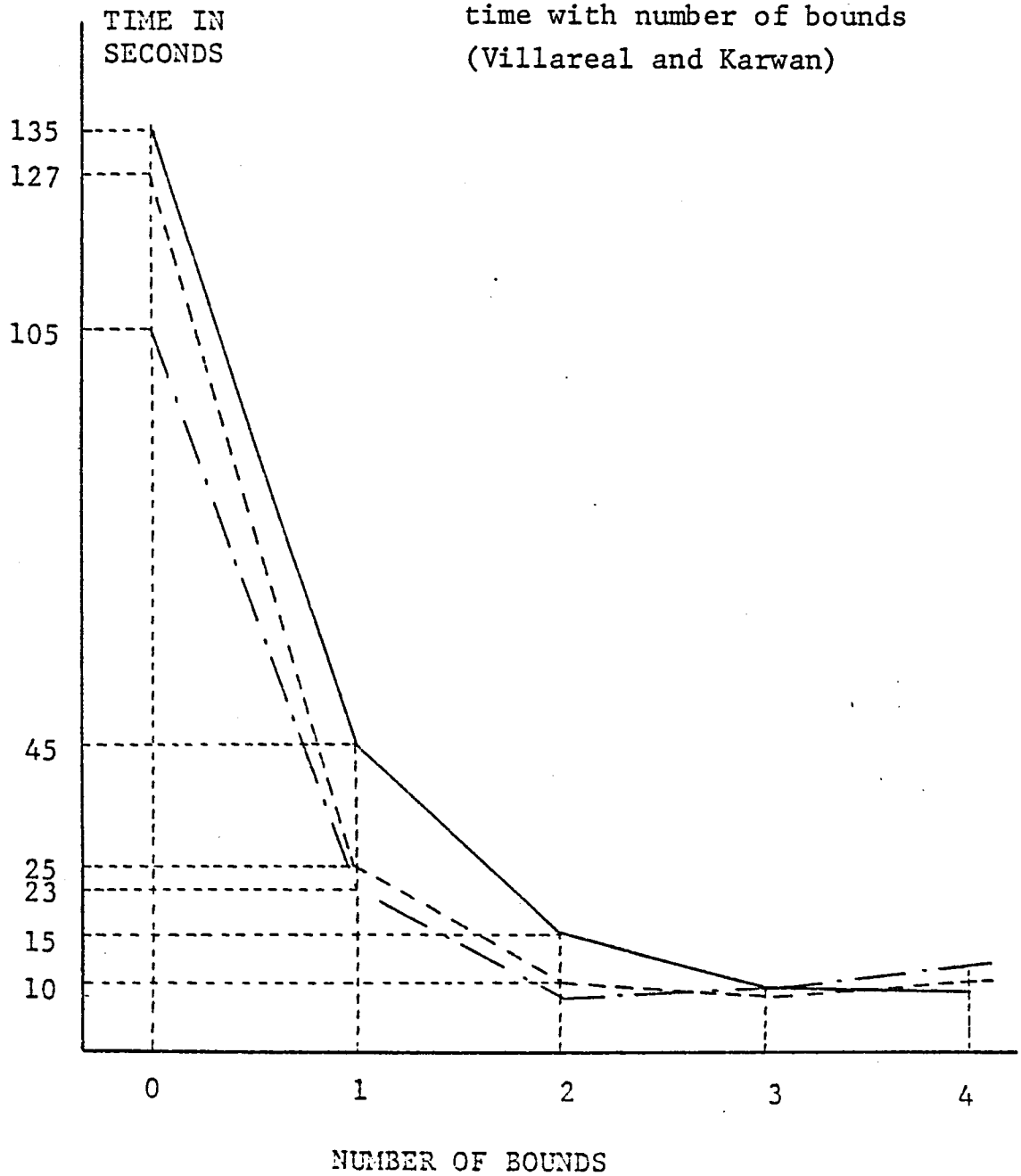
has to be paid for calculating these bounds.

It is interesting to see how the solution time varies with the number of bounds performed. Villareal and Karwan analyze this problem for a multiobjective function (p=2) and a size of 10 variables and 10 constraints.

Different problems were run several times, the first one with no bounds, the second one with two lower bounds, the third with three and the fourth with four lower bounds. The decrease in solution time between the first run and the second one was of about 93,7%. One more lower bound decreased the time of solution by 27,5% over the previous one, and only in four out of ten problems. A fourth lower bound decreased the solution by only 9,9%. The six remaining problems' time solutions already begin to increase in the moment they perform the third bound.

Then, depending upon the kind of problem, we can draw the typical graph (Figure 1) of the variation of the solution time with the number of bounds. There will be a point for which the solution time begins to increase.

Figure 1. Variation of solution time with number of bounds (Villareal and Karwan)



Finally, computational experience is reported. In using the hybrid dynamic procedure, the heuristic developed by Loulou and Michaelides [36] was applied to obtain sets of lower bounds to the original problem. This proved to be very useful since many of the bounds generated turned out to be efficient solutions. The sets of upper bounds for the residual problems, were determined by setting each of the remaining variables to their upper bounds. The reduction in the solution time of the problem was proved significant. In particular, for those with b-value equal to 0.75 times the sum of the associated row coefficients.

Villarreal and Karwan end up saying that the interactive branch and bound approach gives the best computational time.

1.3.3 Other Algorithms for M.P.P.

Zionts and Wallenius [48] and Zionts [49] present theoretical results, algorithms and computational experiences for the versions of (P) where F is a finite discrete set given

explicitly.

Two methods are offered: one is based upon cutting plane theory, and the other uses branch and bound ideas. The initial step of both procedures is to solve the linear relaxation of the problem. If the solution satisfies the integrality constraint, it is the optimal solution and the procedure stops. Otherwise, one must continue until satisfying integrality. The procedures differ only in the way they achieve this goal. Zionts recommends the use of the branch and bound algorithm based upon prior experiences reported with similar algorithms for solving single objective problems. In both procedures it is assumed that the decision maker has a linear additive utility function. This utility function is not entirely or necessarily known.

The technique developed by Zionts is analogous to the work of Zionts and Wallenius, except that the set of decision is discrete, finite and explicitly given. The final result is not a solution but, a collection of solutions that the manager appears to prefer over the others.

The technique consists of two phases. In the first phase an efficient solution is found. Next, an arbitrary set of weights is selected to start and each alternative is evaluated. The one that maximizes the corresponding linear functional is selected and every efficient adjacent decision is generated over it.

The procedure is repeated by posing new trade offs to the decision maker at the current solution. The method converges to the optimal solutions for the unknown overall utility function in a finite number of iterations.

If the true utility function was known, we would use the objective values of the solutions to decide branching rules. However, this is not possible, since we have only the approximation. Under the assumption that the decision maker knows implicitly his preference weights, the decision may be based on his responses to where he would rather continue branching. This is valid since it implies that the best preferred solution will have the greatest utility for the decision maker. For instance, if the solutions x and y are available and x is preferred to y, then $\lambda cx > \lambda cy$, where λ denotes the true

set of preference weights. Notice that $\lambda cx > \lambda cy$ does not necessarily imply that $cx \geq cy$ with at least a strict inequality. However, the preferred solution in this case must be a member of the set of efficient solutions of the problem. As a matter of fact, it must be a strong efficient point.

SECTION 2

A REVIEW OF INTEGER PROGRAMMING FOR A SINGLE OBJECTIVE FUNCTION

Among the most general approaches to the solution of mixed-integer optimization problems is the Branch and Bound method. The Branch and Bound method is nothing more than an intelligently structured search of the space of all feasible solutions. Most commonly, the space of all feasible solutions is repeatedly partitioned into smaller and smaller subsets, "Branching", by a basic tree enumeration method which involves calculating upper bounds and lower bounds, "Bounding", on the objective function in order to be able to find the points in which no further exploration can be profitable, "Fathoming". The utility of the method derives from the fact that only a small fraction of the possible solutions needs actually be enumerated.

As a special case of the Branch and Bound method, we will focus our attention upon the one used to solve binary variable problems, where all the integer variables are required to be 0,1.

2.1 Implicit Enumeration

Implicit enumeration is the name of a class of Branch and Bound algorithms designed specifically for the case in which x is required to be a binary vector.

$$
\begin{aligned}
(P) \qquad & \text{Min} \quad z = cx \\
& \text{s.t.} \quad Ax \geq b \\
& \qquad\; x = 0,1 \ \text{(Binary)}
\end{aligned}
$$

We can also assume, with no loss of generality, that $c \geq 0$, since any x_j with $c_j \leq 0$ can be replaced by $x'_j = 1 - x_j$

Balas [3], characterizes his algorithm as 'additive' because no multiplication or division are required. He applied his additive method to mixed integer programming with zero-one variables, where u is a solution of the problem (P).

The fundamental idea underlying the algorithm runs in the following lines: He starts with the relaxation of the binary constraints, problem (P_o) of solution u_o.

Starting with all n variables equal to 0, the algorithm

consists of a systematic procedure of assigning the value 1 to some of the variables in such a way that, by following some branches of the tree, and after trying a small part of all the 2^n possible combinations, one obtains either an optimal solution or evidence of the fact that no feasible solution exists.

At a stage k, the solution u^k is then uniquely determined by the set

$$S_k = \left\{ j : x_j^k = 1 \right\}$$

Balas bases his computation on trying to find the new vector to be introduced into the basis at a stage k + 1, (BRANCHING). According to certain rules, he chooses the next variable to be introduced as a constraint from the set of improving vectors, N_k, of the solution u_k. This set, N_k, is essentially the set of points which being still free, j S_k, if introduced into the problem, the value of the objective function won't hit the ceiling $\bar{z}_k$ for u_k (BOUNDING).

Whenever a solution u_s is reached, only the improving vectors for that solution are considered for introduction into the basis. Whenever the set of improving vectors for a so-

lution u_s is found to be empty that means there is no feasible solution u_t such that $z_t < z_s$. In such cases we have to take up our procedure from a previous solution u_k.

An essential feature of the Balas algorithm is the testine of the bounding problems for infeasibility. Each linear constraint is examined to see if it can be satisfied by setting some variables to its more favorable value. If,

$$(1) \qquad \sum_{j \in N_k} a_{ij}^{-} \leq y_i^k \qquad\qquad a_{ij}^{-} = \begin{cases} a_{ij} & \text{if } a_{ij} \leq 0 \\ 0 & \text{if } a_{ij} > 0 \end{cases}$$

does not hold for every i, no completion of S_k can satisfy constraint i and we can fathom this point.

The Balas Algorithm terminates when a solution has been reached, for which a) there are no improving vectors for any u_k, b) inequality (1) does not hold for any k such that $N_k \neq 0$ and c) we can not back up any more.

Although Balas first put the fundamentals of the algorithm it was Geoffrion who clearly defined the implicit idea of Balas [20].

A partial solution S_k is defined as an assignment of binary values to a subset of the n variables. Any variable not assigned a value by S_k is called "free".

A completion of a partial solution, is defined as a solution that is determined by S_k, together with a binary specification (0 or 1) of the values of the free variables.

Using Garfinkel and Nemhauser [18], notation, we can decompose the set of assigned variables into two different sets:

$$\left\{ S_k^+ = \quad j : j \in S_k \text{ and } x_j = 1 \right\}$$

$$\left\{ S_k^- = \quad j : j \in S_k \text{ and } x_j = 0 \right\}$$

$$\left\{ F_k = \quad j : j \notin S_k \right\}$$

Then, for a given partial solution S_k we can determine the "best" feasible completion (the one that minimizes cx among all feasible completions of S_k). If such a best feasible completion is better than the best known feasible solution, then it replaces the latter in store. The best feasible solution found so far is called the "ceiling" by Balas or

the "incumbent" by Geoffrion. We will call it the upper bound $\bar{z}_o$ if we are in a minimization, and the lower bound $\underline{z}_o$ if we are in a maximization. Another possibility is that we may be able to determine that S_k has no feasible completion better than the incumbent. In either case, we shall say that we can fathom this point.

$$(P_k) \qquad \text{Min} \quad z_k = \sum_{j \in F_k} c_j x_j + \sum_{j \in S_k^+} c_j$$

$$\text{s.t.} \quad \sum_{j \in F_k} a_{ij} x_j \geq b_i - \sum_{j \in S_k^+} a_{ij} = y_i^k$$

We want to find the best feasible completion of S_k. That is trivial, just take $x_j = 0$, or 1, for each free variable according to the sign of his coefficient. For convenience, we assumed that $c \geq 0$, so that each free variable may be taken to be zero. We will call the value of z_k, obtained as described, a bound of z_k, $\underline{z}_k$ in a minimization and $\bar{z}_k$ in a maximization.

This value can be feasible, in that case this bound is then the best feasible completion of S_k. As the computations proceed, the value of the incumbent feasible solution gives

a hopefully good upper bound $\bar{z}_o$ on the optimal value of (P), that can be used as indicated below. Until the first feasible solution has been formed we take $\bar{z}_o = \infty$

If the best completion is not feasible, we do nothing further to find the best feasible completion. Instead, we attempt to determine that no feasible completion of S_k is better than the incumbent. If this is actually the case, then it must be impossible to complete S_k so as to eliminate all of the infeasibilities of x_k and yet improve upon $\bar{z}_o$. To demonstrate this impossibility, it is clearly sufficient to contemplate non-zero binary values only for the variables in the set

$$T_k = \left\{ j \in F_k : cx_k + c_j < \bar{z}_o \quad \text{and} \quad a_{ij} > 0 \quad \text{for some } y_i^k < 0 \right\}$$

because in order to give a value of 1 to some free variable not in T_k would either lead to a higher value than $\bar{z}_o$ or would not contribute to diminishing an infeasibility of x_k (we have made use of our assumption that $c \geq 0$). Hence, if T_k is empty then there could be no feasible completion of S_k that is better than the incumbent, and the point is fathomed.

In fact, the set T_k of Geoffrion is equivalent to the N_x set of improving vectors of Balas.

It is also easy to see that the same conclusion holds for the fathoming for infeasibility case of Balas

$$y_i^k + \sum_{j \; T_k} \text{Max } (0, a_{ij}) \leq 0$$

for some i such that $y_i^k < 0$; for then there could be no way to select free variables so as to eliminate infeasibility.

For the augmentation mechanism, one choice is to augment S_k by one variable from T_k. As the same as Balas, the variable that leaves the least amount of total infeasibility in the next x_k in the sense of making

$$\sum_{i=1}^{m} \text{Min } (y_i^k + a_{ij}, \; 0)$$

an algebraic maximum, over all $j \in T_k$.

As we have seen, Geoffrion is dealing with two kinds of

We must realize that the value of the optimal solution is going to be between these two bounds and depending upon if we are in a minimization or in a maximization, these bounds are exactly the opposite of each other in the way that we have to find them.

For instance, in a maximization, an upper bound $\bar{z}_j \geq z_j^*$ may be calculated by taking each free variable to be 1 ($c \geq 0$). In a minimization a lower bound $\underline{z}_j = z_j^*$ may be calculated by taking each free variable to be 0.

Again, the fathoming cases are

(a) $\bar{z}_k = \underline{z}_k$

(b) $\underline{z}_k \geq \bar{z}_o$ (minimization)

Note that (a) occurs when x_k is feasible to P_k, and no better solution can be found. When case (b) occurs, no successors of k can yield a solution that improves on the best known solution to P. Note also, that the case $T_k = \emptyset$ is included in case (b), since $\underline{z}_k = \infty$.

2.2 Additional Tests for Implicit Enumeration

An interesting result due to Zionts [47] is the use of his Extended Geometric Definition Method in the Balas algorithm. The E.G.D.M. is a means of computing and recomputing upper and ower bounds on both primal and dual variables of linear programming problems between simplex method iterations.

For our problem

$$\text{Min} \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j=1}^{n} a_{ij} x_j + x_{n+i} = b_i \qquad i=1,\ldots,m$$

Zionts finds the following bounds:

-A LOWER BOUND for a SLACK variable x_{n+i} is given by

$$h_{n+i} = \text{Max} \left\{ 0,\ b_i - \sum_{k=1}^{n} a_{ik}^{-} \right\}$$

-An UPPER BOUND for the same SLACK variable will be

$$u_{n+i} - b_i = \sum_{k=i}^{n} a_{ik}^{-}$$

where

$$a_{ij}^{+} = \begin{cases} a_{ij} & \text{if } a_{ij} > 0 \\ 0 & \text{if } a_{ij} \leq 0 \end{cases} \qquad a_{ij}^{-} = \begin{cases} a_{ij} & \text{if } a_{ij} \leq 0 \\ 0 & \text{if } a_{ij} > 0 \end{cases}$$

-A LOWER BOUND for a ZERO-ONE integer variable will be

$$h_j \begin{cases} \text{Max}\left\{0,\ (b_i/a_{ij}) - (1/a_{ij})\left(\sum_{k=j} a_{ik}^{+} + u_{n+i}\right)\right\} & \text{for } a_{ij} > 0 \\ \text{Max}\left\{0,\ (b_i/a_{ij}) - (1/a_{ij})\left(\sum_{k=j} a_{ik}^{-} + h_{n+i}\right)\right\} & \text{for } a_{ij} < 0 \end{cases}$$

-An UPPER BOUND for the same BINARY variable is

$$u_j \begin{cases} \left\{(b_i/a_{ij}) - (1/a_{ij})\left(\sum_{k=j} a_{ik}^{-} + h_{n+i}\right)\right\} & \text{for } a_{ij} > 0 \\ \left\{(b_i/a_{ij}) - (1/a_{ij})\left(\sum_{k=j} a_{ik}^{+} + u_{n+i}\right)\right\} & \text{for } a_{ij} < 0 \end{cases}$$

And in the case of equality constraints $u_{n+i} = h_{n+i} = 0$. Note that the application of these bounds can be made for partial solutions by computing y_i^k instead of b_i. Then of course, only free variables are considered in the computations.

By using these results to generate upper and lower bounds

on the variables we can apply them to the Implicit Enumeration Method.

-If a lower bound greater than zero, but less than or equal to one, is found for some variable x_k, then this integer variable is implied to be one in all continuations of the present partial solution.

-If a lower bound greater than one is found for some variable x_k, then there is no feasible continuation.

-If an upper bound less than one, but not less than zero is found for some variable x_k, then x_k is implied to be zero in all continuations of the present partial solution.

-If an upper bound less than zero is found for some variable x_k, then there is no feasible continuation.

-If all upper bounds are at least one and all lower bounds are at most zero, then no tighter bounds are available.

Actually, the calculations of the upper and lower bounds

can be simplified tremendously by the elimination of tests which cannot occur, and avoiding repetitious calculations, as we will explain later.

Another important point in the implicit enumeration algorithm is the choice of the partitioning variable. C. Lemke and K. Spielberg [35], analyze the computational advantage of some modifications of Balas' algorithm, in particular the issue of the preferred row and the preferred variables.

In order to choose the next variable to be fixed, they constructed a preferred set of variables for each row $T_k(i)$, and defined the preferred row as the row for which $T_k(i)$ has the minimal number of entries. In the tree search a preferred row is associated with each point of the tree when that point is reached for the firss time. Then, and also at every subsequent return to this point, the preferred set for this row is established and a forward branch is selected from the indices in this set only.

They also applied the concept of "complete enumeration", of which we will now give an example.

For each constraint i, $y_i^k < 0$, find the Gomory cut. This is nothing more than taking all variables with positive coefficients to zero.

$$-x_1 +3x_2 -5x_3 -x_4 +4x_5 \leq -3$$

$$-x_1 -5x_3 - x_4 \leq -3$$

This tells us that some of these three variables have to be set at one. Furthermore, we can determine which of the three it is.

Put all variables in order of decreasing coefficient and systematically fix $x_j = 1$, beginning with the one which has the greatest coefficient.

$$-x_1 -x_4 -5x_3 \leq -3$$

$$-x_4 -5x_3 \leq -2$$

$$-5x_3 \leq -1$$

$$x_3 \geq 1/5$$

What we have found is only a lower bound for variable x_3;

the same we would have obtained using Zionts' test. Being as how $h_3 = 1/5 > 0$, this means that x_3 has to be set at one.

They have found that this procedure substantially reduces the number of points to be considered, as well as the computing time.

As we will soon see, the solution time increases exponentially with the number of variables. This is a reason why the Zionts test for finding upper and lower bounds in variables, explained in Part Two of this section, have such great computational advantages.

In fact, any branching rule taking into account feasibility, will lead us to choose the same variable as in Zionts' test, but the advantage of one in respect to the other is clear. Ziont's test will tell us that a variable has to be set at a specific value. Instead, the branching rule will recommend us to take this variable to the same value. Afterwards, when we check the other branch of the tree, where this variable takes the opposite value, we will see that this branch is infeasible, but we will have lost more computational time.

C. Lemke and K. Spielberg, in their complete enumeration of each constraint, are doing the same as Zionts. In fact, they find the lower bound for only one variable, the one with the most negative coefficient. This suggests to us that it is not necessary to find all the lower and upper Zionts bounds for all the variables, but rather, for only some preselected variables that we will now try to identify.

Consider the problem

$$\text{Max} \quad cx$$

$$\text{s.t.} \quad Ax \leq b$$

$$x = 0,1$$

where all $c_{ij} \leq 0$. Then, to maximize cx we would like to set all variables at 0. This normally won't give us feasibility. So, our goal is to look at those constraints with b_i 0, and find which variables have to be set at 1, in order to make the problem feasible ($b_i > 0$), because in this moment, the implicit enumeration algorithm will stop, taking all the rest of the variables to 0.

So, we only have to analyze constraints $i \in Q_k$, where Q_k is the set of $i : y_i^k < 0$. Among the constraints, we begin by analyzing the variables in the set R_k^-, the set of $j : a_{ij} < 0$.

A lower bound in this case will be

$$h_j = \text{Max} \left\{ 0, \underset{>0}{(b_i/a_{ij})} + \underset{<0}{(1/a_{ij})} \sum_{k \neq j} a_{ik}^- \right\}$$

So, we want

$$b_i + \sum_{k \neq j} a_{ik}^- < 0 \Rightarrow \sum_{k \neq j} a_{ik} < |b_i|$$

and that $(b_i + \sum_{k \neq j} a_{ik}^- / a_{ij})$ be as big as possible. To

To achieve this, we need to have $\sum_{k \neq j} a_{ik}^- > 0$, as small as possible, and $a_{ij} < 0$, as small as possible (the most negative number).

To find which variable to test, we can continue as follows: Reindexed constraint i so that the coefficient a_{ij} be in non-decreasing order.

We can define $T_i(m) = \sum_{j \neq m} \bar{a}_{ij}$, and find $T_i(m)$ for each variable $m \in R_k^-$, beginning with the $q = \underset{j \in R_k^-}{\text{Max}} |a_{ij}|$, (the first variable in the reindexed constraint i).

We will find p, such that $T_i(p) < |b_i| < T_i(p+1)$, and test all the variables between p and $q = \underset{j \in R_k^-}{\text{Max}} |a_{ij}|$.

A simplification would be to test only for variable q. If for q, $T_i(q) = \sum_{j \neq q} \bar{a}_{ij} > |b_i|$, we know that not only $h_q < 0$ but rather any other variable j such that $|a_{ij}| < |a_{ip}|$ will have

$$\sum_{k \neq j} \bar{a}_{ik} > \sum_{k \neq q} \bar{a}_{ik} \quad \Rightarrow \quad \sum_{k \neq j} \bar{a}_{ij} > |b_i|$$

So, it will have $h_j < 0$.

Unfortunatly, this is only a necessary condition, because if it is true, it is possible that $h_q < 0$, and another variable j have $h_j > 0$.

On the other hand, we dont need to find any upper bound

for any variable in R_k^-, because this will tell us that some variables have to be set to 0 and this is automatically done by the implicit enumeration method.

We would also like to find a lower bound for the variables $j \in R_k^+$, the set of $j : a_{ij} > 0$, to see if it is necessary to set some of them at one:

$$h_j = \text{Max} \left\{ 0, \; (b_i/a_{ij}) - (1/a_{ij}) \sum_{k \neq j} a_{ik}^+ \right\}$$

$$< 0 \qquad\qquad > 0$$

So, we don't have to find any lower bound in this case, because it will always be less than zero. By the same reason as before, we do not need to find any upper bound for these variables.

Thus, we can conclude that complete enuneration, or finding the lower bound for variable $q = \underset{j \in R_k^-}{\text{Max}} |a_{ij}|$, which is the same thing, is a very good approximation to the Zionts test.

2.3 Computational Experiences

The ultimate practical usefulness of any integer programming algorithm depends on the critical question: "How fast do soultion times increase with the problem size?" The number of variables is perhaps the main determining factor for implicit enumeration algorithms, since the number of possible solutions of (P) is 2^n.

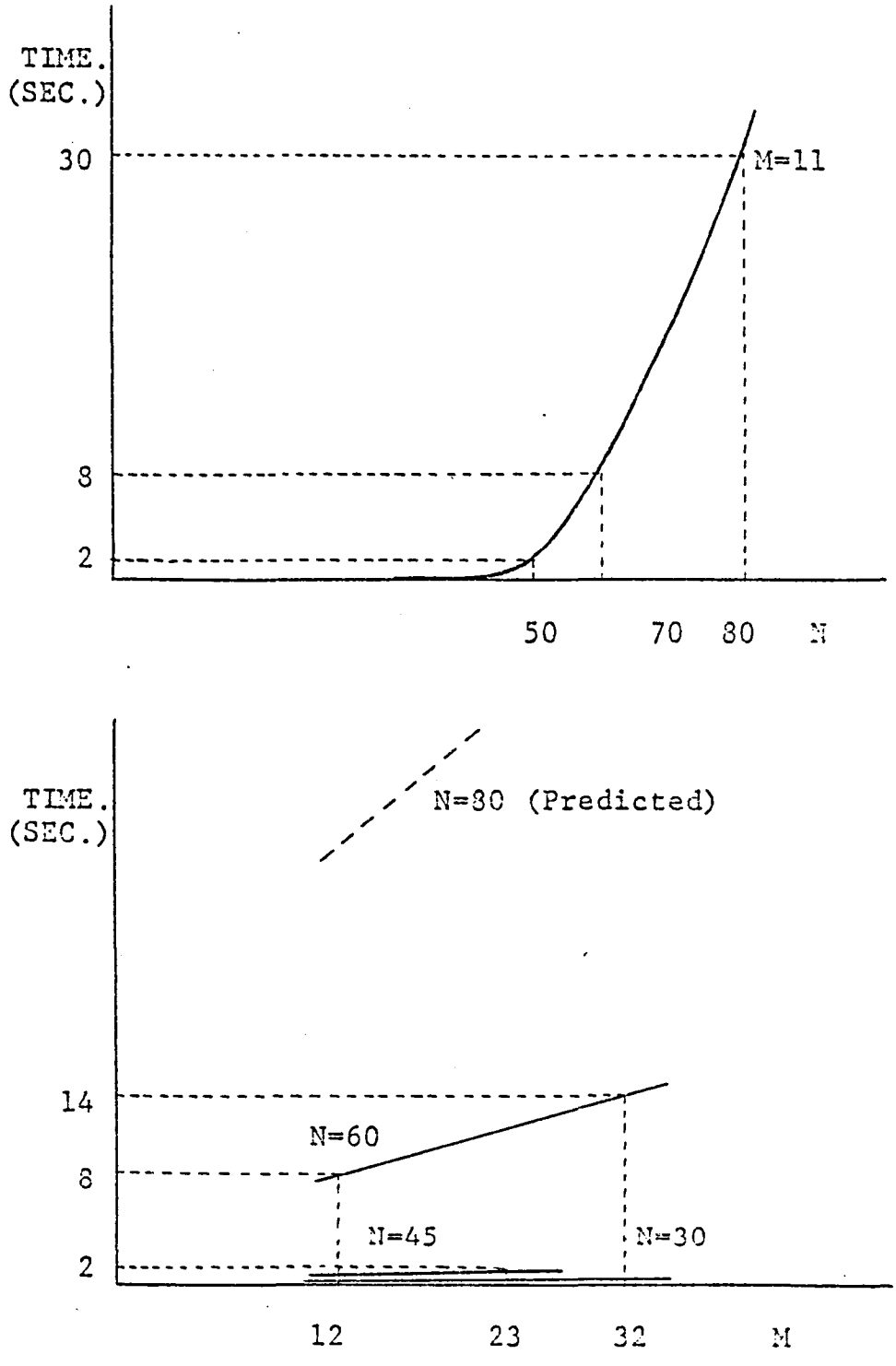
We are going to measure the problem size by two variables: n: the number of variables, and m: the number of constraints. Then we define the problem size by n,m.

The inicial Balas algorithm modified by Fleischmann [16], gives us some interesting results. As we can see in the next graph, for the type of problems analyzed by him, while the increase in time solution is more or less linear as we increase m (keeping n constant), the increase in solution time is very rapid when we increase n (keeping m constant).

Two facts are interesting tc point cut:

-The increase in solution time with n for a given m, is produced in a very short period of n.

FIGURE 2. FLEISCHMANN: COMPUTATIONAL EXPERIENCE WITH THE BALAS ALGORITHM.



-The increase in solution time with m for a given n, (slope of the line), is increasing with n.

Another important issue is that the solution time is going to depend very much not only upon the size of the problem (n,m) and of course, the type of computer used, but also on the kind of constraints analyzed. Fleischmann analyzed a kind of problem of m-1 constraints of the type

$$\sum_{j \in S} x_j \leq 1$$

which gives him the possibility to have good computational results at (50,11) and (44,23), (solution time on the order of one minute). But even that kind of problem blew up at (60,32), 13.8 minutes and (80,11), 30.6 minutes using an IBM 7094.

Now let's analyze the computational impact of the different tests studied before.

Geoffrion [21], analyzed several problems with surrogate constraints and got very good computational results. Each problem was run twice, once skipping the step so that no sur-

rogate constraints were ever computed, and another with the step fully implemented so that an attempt was made to compute a new surrogate constraint each time. The average reduced solution time was about 80%. In fact, problems such as Fleischmann's were reduced by 99.5% Problems on the order of greater than 10 minutes were reduced on the order of seconds. The (80,11) Fleischmann problem was reduced to 0.03 minutes.

These results indicate that the use of the imbedded linear program of Geoffrion greatly reduces the number of required iterations, but even with this device, the algorithm did blow up in a (74,37) problem. It does not matter, therefore, how improved our algorithm is, there is going to be a moment in which the solution time is going to blow up, and normally within a very short range of n. All the tests already explained effectively decrease the average solution time but these tests are normally good for a specific kind of problem, and maybe for another problem of the same size might blow up

Given that we have seen that one of the drawbacks of the algorithm is the number of variables n, the Zionts [47] test

for bounded variables can give us good reductions in solution times. The number of partial solutions computed was also greatly reduced by an average of about 70%. Although given that Zionts only solved very small problems of size (12,6), we really do not know how this test will perform on large scale problems.

SECTION 3

AN APPLICATION TO THE MULTICRITERIA CASE WITH BINARY VARIABLES

The vector maximization problem in 0,1 variables as stated in the first chapter, can be written in the following form

$$(P) \qquad \text{v. max.} \quad \sum_{j=1}^{N} c^j x_j$$

$$\text{s.t.} \quad \sum_{j=1}^{N} a^j x_j \leq b$$

$$1 \geq x_j \geq 0 \quad \text{INTEGER}$$

Our goal is to find the set of efficient points of problem (P). The approach that we will describe is an appliation of the implicit enumeration algorithm, already analyzed for a single objective in the last chapter, but before beginning this study, let us consider again the auxiliary problem (the unit hypercube) in an attempt to obtain all the possible efficient points.

3.1 The Auxiliary Problem

The problem of the unit hypercube can be stated as follows

$$(P') \qquad \text{v. max.} \quad \sum_{j=1}^{N} c^j x_j$$

$$1 \geq x_j \geq 0 \quad \text{INTEGER}$$

If we find all the efficient points in the unit hypercube (P'), we will find a set EF(A), which can be subdivided into two different sets by only testing feasibility in our original problem (P)

$$EF(A) \begin{cases} \nearrow \text{LB(A)} & \text{FEASIBLES IN (P)} \\ \searrow \text{UB(A)} & \text{INFEASIBLES IN (P)} \end{cases}$$

We call those sets (A), because they belong to the AUXILIARY problem (unit hypercube P').

The set EF(A) is the set of all efficient points of the

auxiliary problem. The set LB(A) consists of those efficient points that are feasible in our original problem (P) and therefore are also efficient in it. They will form an initial lower bound set (LB) that afterwards we will be able to use in the implicit enumeration method. The set UB(A) consists of those efficient points that are not feasible in our problem (P) and therefore will dominate the rest of the efficient solutions that we need to find. They will form an upper bound (UB) set which dominates all the efficient points not included in the LB(A) set.

If we could find the directly dominated points of each element of the UB(A) set, we would get the following kinds of points:

-Points dominated by the LB(A) set. We discard them.

-Points not dominated by the LB(A) set. We test feasibility in (P).

-Points feasible in (P). They may or may not be efficient, but we include them in the LB(A) set. When no infeasible points remain we test dominance between the elements of this set and obtain the final set of efficient points of (P).

-Points infeasible in (P). In order to find their dominated points, we have to repeat the process. Eventually we reach a moment in which no infeasible points remain.

Two problems need to be solved for this approach:

-Find all the efficient points in the unit hypercube (P') or EF(A).

-Find a method to locate all the dominated points of a certain element of the set UB(A).

Solving these two problems will require as we have seen in Bitran's paper, a lot of computations, but we can still use these concepts to start the second approach with good lower bounds that can improve substantially the fathoming step.

3.1.1 Finding Efficient Points over the Unit Hypercube

If all the objective functions contain all the variables then, we can find at most, p efficient points, by only maximizing each of the p objective functions.

In this maximization we only need to take a variable to

be either 0 or 1 depending upon the sign of its coefficient. For each objective, we set x_j to 1 if $c_{ij} > 0$, and to 0 if $c_{ij} < 0$. In this case we find between 1 and p efficient points depending upon the coefficients of every objective function.

If some of the objective functions contain only some of the variables, then, in general we can find more than p efficient points. This is because when we have to maximize a function of less than n variables we will have some unfixed variables. In order to fix the remaining variables we simply find another objective function that contains those variables and maximize it. We continue this procedure until we have fixed all variables. It is clear that depending upon the objective function selected we can find different efficient points, so we must check all possible combinations and, finally test dominance among all the points found.

This may be accomplished in the following way: We know that the variables fixed at 0 or 1 in the maximization of a particular objective are not going to change their status. Therefore, we have to worry about the remaining variables. To do this, we check the coefficients of a particular not

fixed variable, over all the objective functions and come up with three different cases:

-If all the coefficients are positive, then we only need to set this variable to 1.

-If all the coefficients are negative, then by the same token, we will set this variable to 0.

-If the coefficients are positive and negative, then we need to set the variable to 0 and 1.

After doing this for every non-fixed variable, and maximizing each one of the objective functions we will obtain a set of vectors whose components are zeros and ones with the peculiarity that among its members we have all the efficient points of problem (P').

To find all the efficient solutions we have to do just two more things:

-First, eliminate all the vectors that have the same components, because probably some vectors will be repeated.

-Second, find the values of each of these vectors in the multicriteria space and test dominance among them via pairwise comparisons.

The result will be all the efficient points in the unit hypercube. It is interesting to note that the value in the multicriteria space of a specific objective function will vary between a certain upper value found by the maximization of this particular objective function and a lower bound that will be the minimum value that takes the same objective among the rest of the maximizations.

That is to say, for a particular objective k, this value in the criteria space will vary between the value of the maximization of k and the minimum of all the maximizations of i, $i \neq k$, $i = 1,\ldots,p$.

Having found EF(A), we will see how in the next approach (the implicit enumeration algorithm) we can use the two sets LB(A) and UB(A) for fathoming purposes.

3.2 An Implicit Enumeration Algorithm

As we have seen in the last chapter the implicit enumeration algorithm is based on a number of concepts that now will be analyzed for the multiobjective case.

The algorithm is basically the same as in a single objective problem. The separation at stage k is determined by choosing a particular variable x_j not selected previously along the path P_k from v_o to v_k. The path P_k corresponds to an assignment of binary values to a subset of the variables.

If we denote the set of assigned variables as W_k, and we let:

$$S_k^+ : (j : j \in W_k \text{ and } x_j = 1)$$
$$S_k^- : (j : j \in W_k \text{ and } x_j = 0)$$
$$F_k : (j : j \notin W_k)$$

Then we can consider the problem at stage k

$$\text{(MP)} \quad \text{Max} \quad z_k = \sum_{j \in F_k} c^j x_j + \sum_{j \in S_k^+} c^j$$

$$\text{s.t.} \quad \sum_{j \in F_k} a^j x_j \leq b - \sum_{j \in S_k^+} a^j = s_i^k$$

In the development of the algorithm, we need to analyze a number of concepts such as Bounding, Fathoming, Branching, etc. which shall now be considered.

3.2.1 Bounding Procedures

There are two kinds of bounds that we must analyze: LOWER and UPPER bounds, and the first thing that we need to do is clearly define both of them.

LOWER BOUND SET: Each element is either efficient or is dominated by at least one efficient solution of the problem.

UPPER BOUND SET: Each efficient solution of the problem is dominated by at least one member of the set.

Next, let us discuss the various ways that these bounds may be found.

LB. LOWER BOUNDS.

LB1 THE AUXILIARY PROBLEM.

A first set of lower bounds can be found in the auxiliary problem (P') as we have seen in section 3.1. Finding the set of efficient points in the unit hypercube EF(A) we can obtain a first set of lower bounds by testing feasibility in problem (P).

The set of points of EF(A) which are feasible in (P), will form a lower bound set LB(A) of (P).

LB2 λ's METHOD

By solving the next problem (P_λ), we can find another set of lower bounds:

$$(P_\lambda) \qquad \text{Max} \quad \sum_{i=1}^{p} \lambda_i \, f_i(x)$$

$$\text{s.t.} \quad \sum_{j=1}^{p} a^j \, x_j \leq b$$

$$1 \geq x_j \geq 0 \quad \text{INTEGER}$$

where $f_i(x) = \sum_{j=1}^{N} C_{ij} \, x_j$

Any set of positives values of λ_i will give us a lower bound that some times may be an efficient solution to our problem. The number of points to generate will depend on the computational advantages achieved by using the extra points.

LB3 HEURISTICS

Another set of lower bounds could be obtained by compu-

ting feasible solutions to the (P_λ) problem just described. These feasible solutions may be found using heuristics such as those described in Toyoda [43], and Loulou and Michaelides [36].

LB4 EFFICIENT SOLUTIONS

In general, we must not forget that any solution to the original problem which is efficient in (P), is a good lower bound for the problem. Therefore, in this procedure, when we find another efficient solution to the problem, we can add it to the lower bound set.

UB UPPER BOUNDS

Different sets of Upper Bounds may be formed by the following methods:

UB1

Obtain the solution to p different problems of the following form, where the objective function is one of the p original objectives of (P).

$$(P_p) \qquad \text{Max} \quad \sum_{j \epsilon F_k} C_{ij} \, x_j$$

$$\text{s.t.} \quad \sum_{j \epsilon F_k} a^j \, x_j \leq s$$

$$1 \geq x_j \geq 0 \quad \text{INTEGER}$$

We will form a vector whose i^{th} position or component corresponds to the value of the soution to the problem with the i^{th} objective function. UB(F). This procedure has to be executed for each stage of the problem, and since this upper bound is done only for the free variables, we have to add it, component by component to the value of z at this stage k. So, the value of the upper bound of problem (P), $\bar{z}_k$, at stage k, is

$$\bar{z}_k = z_k \oplus UB(F)$$

UB2

Simpler sets of upper bounds con be obtained by relaxing the problem as follows:

$$\text{Max} \quad \sum_{j \epsilon F_k} C_{ij} \, x_j$$

$$\text{s.t.} \quad 1 \geq x_j \geq 0 \quad \text{INTEGER}$$

If we divide the free variables set at stage k, F_k, into two sets, for each objective function p, we can find:

$$F^+_{ki}: \left\{ j : j \in F_{ki} \quad \text{and} \quad C_{ij} \geq 0, \ \forall i \right\}$$

$$F^-_{ki}: \left\{ j : j \in F_{ki} \quad \text{and} \quad C_{ij} \leq 0. \ \forall i \right\}$$

Then the upper bound set UB(F) in this case, is very simple to find: We set x_j to 1 if $j \in F^+_{ki}$, and to 0 if $j \in F^-_{ki}$, for each objective function.

Then, the upper bound set of our problem (P) will be

$$\bar{z}_k = z_k \oplus UB(F) = \sum_{j \in S^+_k} C^j + \sum_{j \in F^+_{ki}} C^j \qquad i = 1,\ldots,p$$

Where z_k is the solution of the problem at stage k, amd UB(F) is the upper bound of the residual problem at stage k.

UB3

Another way to find an upper bound set is by solving p problems of the form:

$$\text{Max} \quad \sum_{j \in F_k} C_{ij} \, x_j$$

$$\text{s.t.} \quad \sum_{j \in F_k} A_{ij} \, x_j \leq s_i$$

$$1 \geq x_j \geq 0 \quad \text{INTEGER}$$

where the constraint set inforced is any one of the constraints of the original set. The bounding procedure would then require the solution of a serie of different Knapsack problems.

UB4 DUAL PROBLEM

From the solution of the dual problem associated with the linear relaxation of (P)

$$(D) \qquad \text{Min} \left\{ e^t w + s^t u \right\}$$

$$\text{s.t.} \quad w I + u A \leq C_i$$

$$w, u \geq 0$$

where C_i corresponds to the i^{th} objective function, w and u are dual variables associated with the upper bound and resource constraint A respectively, and e^t is the row vector of 1's.

The solution to this problem obviously exists, since $s \geq 0$ and it is attained at one of the extreme points of the set of solutions.

Let the set of extreme points for a given objective function C_i, be denoted by Ω_i. Then the i^{th} component of the upper bound vector UB(F) at stage k, to be denoted by $UB_i^k(F)$ is given by

$$UB_i^k(F) = \text{Min} \left\{ e^t w^o + s^t u^o \right\} \quad w^o, u^o \in \Omega_i$$

For this relationship we have

$$e^t w^o + s^t u^o \geq UB_i^k(F)$$

Then, any basic feasible dual solution can be used as an upper bound for the particular objective function value at stage k.

UB5

Other upper bound sets can be found by relaxing the constraints of problem (P) and keeping the integrality constraints, or viceversa, by relaxing the integrality constraints.

$$(P_{ra}) \qquad \text{v-max} \quad \sum_{j \in F_k} c^j x_j$$

$$\text{s.t.} \quad \sum_{j \in F_k} a_{ij} x_j \leq s_i$$

$$1 \geq x_j \geq 0 \quad \text{INTEGER}$$

and (P_{ri})

$$\text{v-max} \quad \sum_{j \in F_k} c^j x_j$$

$$\text{s.t.} \quad \sum_{j \in F_k} a^j x_j \leq s$$

$$1 \geq x_j \geq 0$$

UB6 LAGRANGIAN RELAXATION

Another upper bound vector can be found through a Lagrangian relaxation, by solving the following problem at stage k.

$$(P_\lambda) \qquad \text{Max} \left\{ C_i x - \lambda(Ax - s) \right\}$$

$$\text{s.t.} \quad 1 \geq x \geq 0 \quad \text{INTEGER}$$

where λ denotes the lagrangian multipliers and Ax represents $\sum_{j \in F_k} a^j x_j$

The associated dual problem is

$$(D_\lambda) \qquad \text{Min} \left\{ \text{Max} \left\{ (C_i - \lambda A)x + \lambda s \right\} \right\}$$

$$\text{s.t.} \qquad 1 \geq x \geq 0 \quad \text{INTEGER}$$

$$\lambda \geq 0$$

We know by duality that $\nu(D) \geq \nu(P)$, being $\nu(.)$ the solution value of problem (.). Rearranging the Dual, we arrive at:

$$(D_\lambda) \qquad \text{Min} \left\{ \lambda s + \text{Max } (C_i + \lambda A)x \right\}$$

$$\text{s.t.} \qquad 1 \geq x \geq 0 \quad \text{INTEGER}$$

$$\lambda \geq 0$$

Geoffrion [22], Glover [25] and Greenberg and Pierskalla [30] have developed results that imply

$$\nu(\bar{P}) \geq \nu(D_\lambda) \geq \nu(P)$$

where $\bar{P}$ is the linear relaxation of problem (P). Geoffrion also suggests that $\nu(\bar{P}) = \nu(D_\lambda)$ in the case in which the problem (P_λ) possesses the following property:

INTEGRALITY PROPERTY: Problem (P_λ), has the integrality

property, if $\nu(P_\lambda) = \nu(\overline{P}_\lambda)$ for all λ.

In our case, it is clear that (P_λ) has the integrality property. So, we only need to solve p linear relaxations of (P).

UB7 SURROGATE RELAXATION

The surrogate relaxation for problem (P), will be

$$(P_\mu) \qquad \text{Max} \quad C_i x$$

$$\text{s.t.} \quad \mu(Ax - s) \leq 0 \quad ; \quad \mu \geq 0$$

$$1 \geq x \geq 0 \quad \text{INTEGER}$$

The associated surrogate dual problem is given as

$$(D) \qquad \text{Min} \left\{ \mu(P_\mu) \right\} \quad ; \quad \mu \geq 0$$

Geoffrion also suggests that

$$\nu(\overline{P}) \geq \nu(D_\lambda) \geq \nu(D) \geq \nu(P)$$

As mentioned earlier, $\upsilon(\overline{P}) = \upsilon(D_\lambda)$ in our case. So, using the value of the surrogate dual for each objective

function, will give us another upper bound vector of our problem.

UB8

Finally, another set of upper bounds is obtained from the unit hypercube. Althoug this UB(A) as we have seen is not like the others already analyzed, an upper bound of the FREE variables. but rather of the entire problem, it will be used in a slightly different way in the following study of fathoming.

3.2.2 Fathoming Criteria

The fathoming criteria that we will study in this section are based fundamentally on the feasibility tests and on the bounding procedures already analyzed. The different tests that we will perform are essentially the same as those studied in the case of a single objective function: Fathoming by FEASIBILITY and fathoming by BOUNDS.

F1 FATHOMING BY FEASIBILITY

We say that a point is fathomed by feasibility when at a given stage of the algorithm, one of more constraints are not satisfied. We know that the matrix of constraints at stage k is given by

$$\sum_{j \epsilon F_k} a_{ij} x_j \leq s_i \qquad i=1,\ldots,m$$

In order to provide a better way of expressing the idea, we will divide the set of free variables at stage k, F_k, into two sets: For each constraint i, we will have

$$F_{ki}^{+} = \left\{ j : j \in F_{ki} \text{ and } a_{ij} \geq 0, \forall i \right\}$$

$$F_{ki}^{-} = \left\{ j : j \in F_{ki} \text{ and } a_{ij} < 0, \forall i \right\}$$

for i=1,...m.

Thus, for each constraint i, we can define t_i as follows. First, set x_j at 0, if $j \in F_{ki}^{+}$, and at 1, if $j \in F_{ki}^{-}$. Let

$$t_i = \sum_{j \epsilon F_{ki}^{-}} a_{ij} x_j \qquad i=1,\ldots,m$$

It is clear that the fathoming by feasibility will take place in the moment that at least one constraint satisties the inequality $t_i > s_i$. In this case, we can delete the point from further consideration.

F2 FATHOMING BY BOUNDS

Two kinds of fathoming by bounds need to be considered:

F2.A

When at a given stage k of the problem, the vector of upper bounds of the objective functions $\bar{z}_k$ is less than or equal to any lower bound vector LB_i of the set of lower bounds LB of our original problem, we can also delete the point.

The upper bound vector $\bar{z}_k$ is equal to:

$$\bar{z}_k = z_k \oplus UB(F) \lesssim LB_i$$

The value of the objective functions plus the upper bound vector on the free variable.

F2.B

Another fathoming case is when at stage k, the vector value of the objective functions z_k, cannot be dominated by

any of the vectors of the upper bound set of the auxiliary problem UB(A), since we know that the rest of efficient points that we need to find must be in the set of points dominated by UB(A).

To achieve this, we have to find the best case in which z_k can appear. Therefore, we will form a lower bound of z_k at stage k, $\underline{z}_k$, by adding to the value of z_k, a lower bound vector on the free variables. This lower bound of the objective functions can be found as follows: Having defined F^+_{ki} and F^-_{ki} for i=1,...,p on page 72, we set a free variable to 0 or 1 depending upon the sign of the objective function coefficient. We set x_j at 0 if $j \in F^+_{ki}$ and at 1 if $j \in F^-_{ki}$.

Therefore

$$\underline{z}_k = z_k + LB(F)$$

$$\underline{z}_k = \sum_{j \in S^+_k} c^j + \sum_{j \in F^-_{ki}} c^j$$

and then, we can fathom the point if the value $\underline{z}_k$ at stage k not dominated by any element of the set UB(A).

That is to say, in order to be able to fathom the point, the relationship ($UB(A) \geq \underline{z}_k$), UB(A) greater than or equal to $\underline{z}_k$ with at least one strict inequality, cannot be verified by any of the vectors of the set UB(A).

This means that at least one component of $\underline{z}_k$ is greater than its respective one in the vector $UB_i(A)$, for all vectors in the set UB(A), although it is not necessary that it be the same component.

3.2.3 Bounding on Variables

As we have seen in the last chapter, the solution time increases exponentially with the number of variables. This can be a reason why the Zionts test for finding upper and lower bounds in variables has such great computational advantages.

In fact, we think that any branching rule taking into account feasibility, will lead us to choose the same variable as in Ziont's test, but the advantage of one in respect to the other is clear. Ziont's test will tell us, a priori,

which variable or variables have to be set to a specific value. Instead, the branching rule will recommend us to take this variable to the same value; afterwards, when we check the other branch of the tree, where this variable takes the alternative value, we will find that this branch is infeasible. The problem will be to know if the computational time spent in the branching is bigger than the one spent in finding the different bounds.

We will now try to find which variables must be treated for bounds. To do this, we begin by dividing the set of constraints in two sets:

$$Q_k^- = \left\{ i : s_i^k < 0 \right\} \quad Q_k^+ = \left\{ i : s_i^k < 0 \right\}$$

and the set of variables for a particular constraint i, in

$$R_k^- = \left\{ i : a_{ij} < 0 \right\} \quad R_k^+ = \left\{ i : a_{ij} > 0 \right\}$$

We begin by analyzing the variables in the constraints in Q_k^-.

BV1 For the variables in the sets Q_k^- and R_k^-.

A. Lower bounds:

$$h_j = \text{Max} \left\{ 0,\ (b_i/a_{ij}) + (1/a_{ij}) \left(\sum_{k \neq j} a_{ik}^- + h_{n+i} \right) \right\}$$

$$\qquad\qquad >0 \qquad\qquad\qquad <0$$

where $h_{n+i} = \text{Max}\left(0,\ b_i - \sum_{k \neq j} a_{ik}^+ \right)$ is always equal 0.

So, we want $(b_i + \sum_{k \neq j} a_{ik}^-)/a_{ij} > 0$, in order to be able to have h_j greater than 0.

A way to find which variable to test is the following: Reindex constraint i so that the coefficients a_{ij} be in non-decreasing order. We can define

$$T_i^k(q) = \sum_{j \neq q} a_{ij}^-$$

and find $T_i^k(q)$ for each variable $q \in R_k^-$ beginning for the variable $q = \underset{j \in R_k^-}{\text{Max}} |a_{ij}|$.

We will find p such that $T_i^k(p) < |b_i| \leq T_i^k(p+1)$. Thus, we have to test the lower bound for all variables between q and p.

B. Upper Bounds.

$$u_j = \left\{ \underset{>0}{(b_i/a_{ij})} - \underset{<0}{(1/a_{ij})} \Big(\sum_{k\neq j} \underset{>0}{a^+_{ik}} + u_{n+i} \Big) \right\}$$

Then, u_j is always greater than 0, for all $j \in R_k^-$ and we will show that it has to be greater than 1. Since

Since, $u_{n+i} = b_i + \sum_{k\neq j} a_{ik}$, then, u_j will be

$$u_j = -\Big(\sum_{k\neq j} a^+_{ik} + \sum_{k\neq j} a^-_{ik} \Big)/a_{ij} \geq 1$$

always greater than one, because $\sum_{k\neq j} a^-_{ik} \geq -a_{ij}$. Thus, we do not have to find any upper bound of any variable of R_k^-.

BV2. For the variables in the sets Q_k^- and R_k^+.

A. Lower Bounds.

$$h_j = \text{Max} \left\{ 0, \underset{<0}{(b_i/a_{ij})} - (1/a_{ij}) \sum_{k\neq j} \underset{>0}{a^+_{ik}} + u_{n+i} \right\}$$

where $u_{n+i} = b_i + \sum_{k\neq j} a^-_{ik}$. Thus, h_j is always less than 0, and therefore we do not need to find any lower bound for any

variable in R_k^+.

B. Upper Bounds.

$$u_j = \left\{ \underset{<0}{(b_i/a_{ij})} + \underset{>0}{(1/a_{ij})} \Big(\underset{k\neq j}{\sum} \underset{>0}{a_{ik}^-} + \underset{=0}{h_{n+i}} \Big) \right\}$$

We can distinguish several cases:

-If $|b_i| > \sum_{k\neq j} a_{ik}^-$, the vertex is infeasible. Backtrack

-If $|b_i| \sum_{k\neq j} a_{ik}^-$, all negatives variables have to be s set to one and all positives to zero.

-If $|b_i| < \sum_{k\neq j} a_{ik}^-$, then, $u_j > 0$.

It is clear, that we do not need to find u_j in the first two cases. However, we do need to find it in the third case, but only for one variable, variable $p = \underset{j\in R_k^+}{\text{Max}} a_{ij}$, and set x_p at zero if $u_p < 1$.

This is clear, because we want the upper bound $u_j > 0$ as low as possible. So, we want a_{ij} as big as possible because $b_i + \sum_{k\neq j} a_{ik}^-$ is fixed.

We can easily see that if for p, $u_p > 1$, it is because $b_i + \sum_{k \neq j} a_{ik}^- > a_{ip}$. For any other variable j such that $a_{ij} < a_{ip}$ is clear that also $b_i + \sum_{k \neq j} a_{ik}^- > a_{ij}$.

If for variable p, $u_p < 1$, it can also work for another variable j with $a_{ij} < a_{ip}$. We can then find the u_j for the next variable q, such that

$$q = \text{Max} \quad a_{ij} \; ; \quad j \neq p; \quad j \in R_k^+$$

an continue finding u_j until the first $u_j > 1$ is found.

BV3 For the variables in the sets Q_k^+ and R_k^-

A. Lower Bounds.

$$h_j = \text{Max} \left\{ 0, \underset{<0}{(b_i/a_{ij})} + \underset{<0}{(1/a_{ij})} \Big(\underset{k \neq j}{\sum}\, a_{ik}^- + \underset{>0}{h_{n+i}} \Big) \right\}$$

where h_{n+i} is equal zero in this case. As we can easily see, h_j is always negative. So, we don't need to find it for any variable in R_k^-.

B. Upper Bounds.

$$u_j = \left\{ (b_i/a_{ij}) - (1/a_{ij})\left(\sum_{k\neq j} a^+_{ij} + u_{n+i} \right) \right\}$$
$$\qquad\quad <0 \qquad\quad <0 \qquad\quad >0$$

where $u_{n+i} = b_i + \sum_{k\neq j} a^-_{ik}$. It is clear, that we do not need to find u_j in this case, because

$$u_j = -\left(\sum_{k\neq j} a^-_{ik} + \sum_{k\neq j} a^+_{ik} \right)/a_{ij} \quad >1$$

The only case that we have to study is when

$$b_i \quad < \sum_{k\neq j} a^+_{ik} + \sum_{k\neq j} a^-_{ik}$$

because when this inequality is not satisfied, the problem is always feasible.

BV4 For the variables in the sets Q^+_k and R^+_k.

A. Lower Bounds.

$$h_j = \text{Max} \left\{ 0, (b_i/a_{ij}) - (1/a_{ij})\left(\sum_{k\neq j} a^+_{ik} + u_{n+i} \right) \right\}$$
$$\qquad\quad >0 \qquad\quad >0 \qquad\quad >0$$

In this case, we do not need to perform this test because

$h_j = -(\sum_{k \neq j} a_{ik}^{-} + \sum_{k \neq j} a_{ik}^{+})/a_{ij}$, always less than zero.

B. Upper Bounds.

$$u_j = \left\{ \underset{>0}{(b_i/a_{ij})} + \underset{>0}{(1/a_{ij})} \underset{>0}{\sum_{k \neq j} a_{ik}^{-}} \right\}$$

As we can see, u_j is always greater than 0. And given that $b_i + \sum_{k \neq j} a_{ik}^{-}$, is always positive, we want a_{ij} as big as possible. So, we will find u_j only for $p = \underset{j \in R_k^+}{\text{Max}}\, a_{ij}$

If $u_p > 1$, it will be also for any other variable q such that $a_{iq} < a_{ip}$. If $u_p < 1$, it could be also true for another variable j with $a_{ij} < a_{ip}$. We can then find the u_j for the next variable q, such that

$$q = \text{Max} \quad a_{ij} \; ; \quad j \neq p; \quad j \in R_k^+$$

and continue finding u_j until the first $u_j > 1$, is found.

A simpler way to do this test, is to avoid this last part and find u_j only for p, because the same procedure will come back to the same row at the next iteration and it will cal-

culate u_j for the new p, which is nothing more than q.

BV5 A simplification for sections BV1.A and BV3.A

A simpler test is the following: Instead of finding $T_i^k(q)$ and testing the lower bound for all variables between q and p, we could do a simpler test only for variable q.

$$q_1 = \underset{j \in R_k^-}{\text{Max}} |a_{ij}| \qquad q_2 = \underset{j \in R_k^+}{\text{Max}} a_{ij} \quad \text{in each case.}$$

For section BV1.A, test only for q_1. If $\sum_{j \neq q} a_{ij}^- > |b_i|$, we know that not only $h_q < 0$, but also that any other variable j such that $a_{ij} < a_{ip}$ will have

$$\sum_{k \neq j} a_{ik}^- > \sum_{k \neq q} a_{ik}^- \qquad \sum_{k \neq j} a_{ij}^- > |b_i|$$

so, any other j will also have $h_j < 0$.

For section BV3.A, test only for q_2 if $\sum_{j \neq q} a_{ij}^+ > b_i$.

A further simplification can be to test for only some constraints. Lemke and Spielberg [35], tested for only one, the one with less entries, because this one, is the more li-

kely to have a lower bound greater than 0. We could find a set of constraints which contains the greatest number of positive (negative) coefficients.

We can construct a similar procedure for our case, and find the constraint with the smallest number of negative variables (positive), if we are in Q_k^- (Q_k^+).

To do this we can simply find the constraint with the highest index J.

$$J_i^- = \frac{\text{\# of total elements in row i}}{\text{\# of negative variables in row i}} \quad \text{for } Q_k^-$$

$$J_i^+ = \frac{\text{\# of total elements in row i}}{\text{\# of positive variables in row i}} \quad \text{for } Q_k^+$$

We have found, then, two constraints, in which we will perform tests 1 and 2 , respectively. (See Summary).

This is not as simple as it might seem. If we analyze tests 1 , we can see that:

-In test 1.1.A, in order to have h_p as big as possible,

BOUNDING ON VARIABLES

SUMMARY

		A. LOWER BOUNDS	B. UPPER BOUNDS
	1.1 R_k^-	FIND h_p : $p=\underset{j\in R_k^-}{\text{Max}} \lvert a_{ij}\rvert$ IFF $\sum_{j\neq p} a_{ij}^- < \lvert s_i^k\rvert$	NO
1. Q_k^-	1.2 R_k^+	NO	FIND u_p : $p=\underset{j\in R_k^+}{\text{Max}} a_{ij}$ IFF $\lvert s_i^k\rvert < \sum_{k\neq j} a_{ik}^-$ IF $\lvert s_i^k\rvert > \sum_{k\neq j} a_{ik}^-$: INFEASIBLE $\lvert s_i^k\rvert = \sum_{k\neq j} a_{ik}^-$: SET ALL VARIABLES: NEGATIVES AT ONE, AND POSITIVES AT ZERO.
	2.1 R_k^-	NO	IF $\lvert s_i^k\rvert \geq \sum_{k\neq j} a_{ik}^+ + \sum_{k\neq j} a_{ik}^-$ ALWAYS FEASIBLE.
2. Q_k^+	2.2 R_k^+	NO	FIND u_p : $p=\underset{j\in R_k^+}{\text{Max}} a_{ij}$

we want the fewest negative variables possible.

-In test 1.2.B, in order to have u_p as low as possible, we want $\sum_{k \neq j} a_{ik}^{-}$ as little as possible.

Analyzing tests 2 , we can see that in test 2.2.A we will need the fewest positive variables possible, because we want $\sum_{k \neq j} a_{ik}^{+}$ as little as possible. To the contrary, in test 2.2.B, we want $\sum_{k \neq j} a_{ik}^{-}$ as small as possible.

Finally , we present a summary of four possible ways of determining bounds on variables, as well as their corresponding block diagrams:

-Diagram number one is a procedure to find all the variables that have to be set to 0 and 1 by bounding in variables. We analyze every constraint and each time that we set a variable, we change RHS's and begin the problem again.

-Diagram number two is the same as number one, except that we will look at all the constraints and set all the variables to their respective values without changing RHS's. At

the end, we must verify if these are contradictory. In this case we say that the problem is infeasible.

- Figure number five shows the same procedure as before, but only taking into account one constraint, the one most likely of being infeasible, measured by the method discussed in Chapter 3. This method is also fathomed by feasibility since it is easy to prove that if the selected constraint is feasible, no constraint can be infeasible.

- Figure number six is a simpler procedure to find only some of all the possible variables that have to be set by bounding. The only difference is that we just test the lower bound for the most negative variable in each constraint. This was coded in fortran IV and put in the main program as subroutine "Feasibility", because as we can see in the flow chart, at the same time that we are setting variables by bounding, we can fathom when the RHS is negative (testing feasibility).

3.2.4 Branching Rules

One of the most controversial aspects of any algorithm based on an implicit enumeration scheme is the branching rule.

FIGURE 3.

FLOW CHART #1

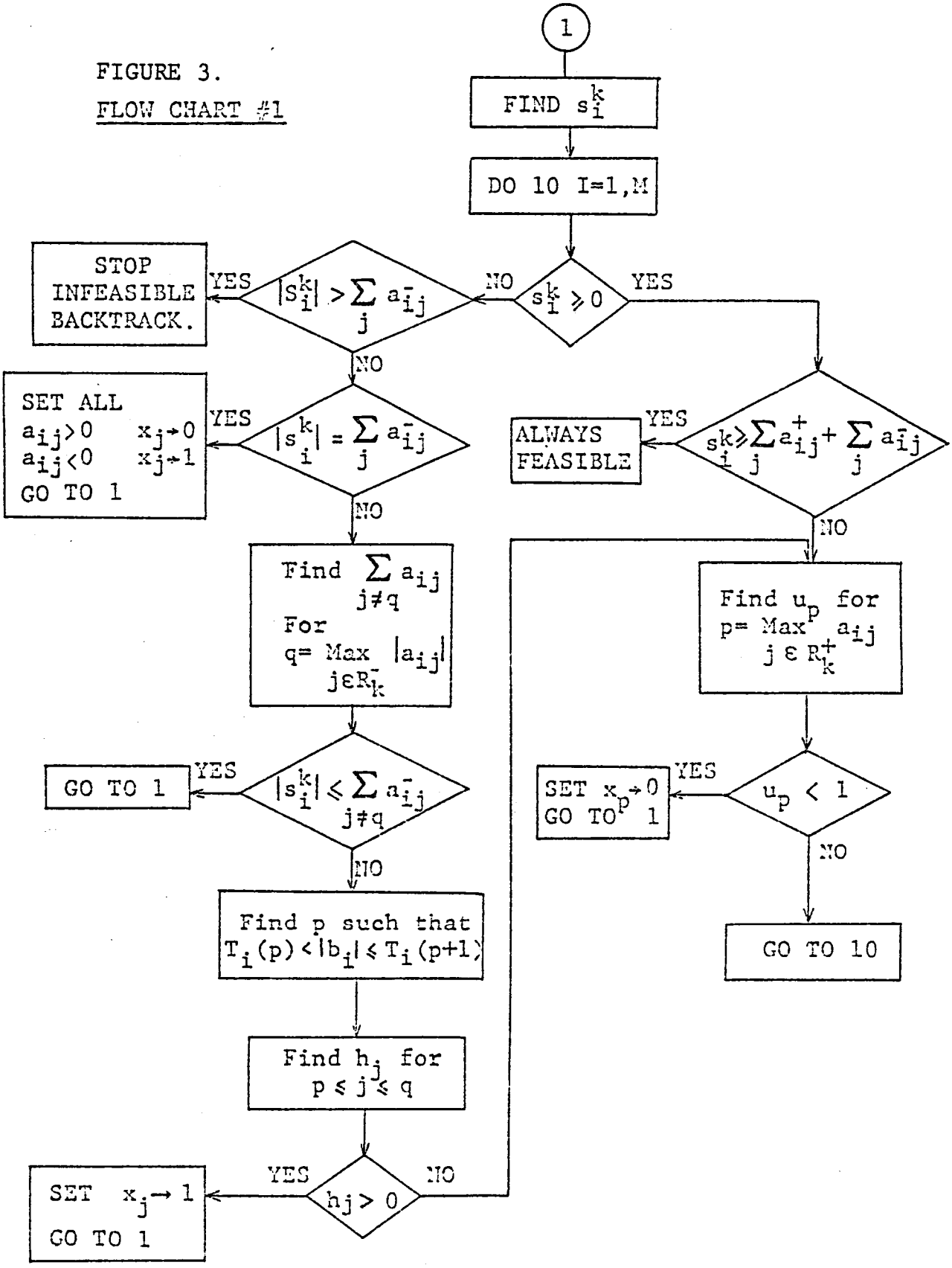


FIGURE 4.

FLOW CHART #2

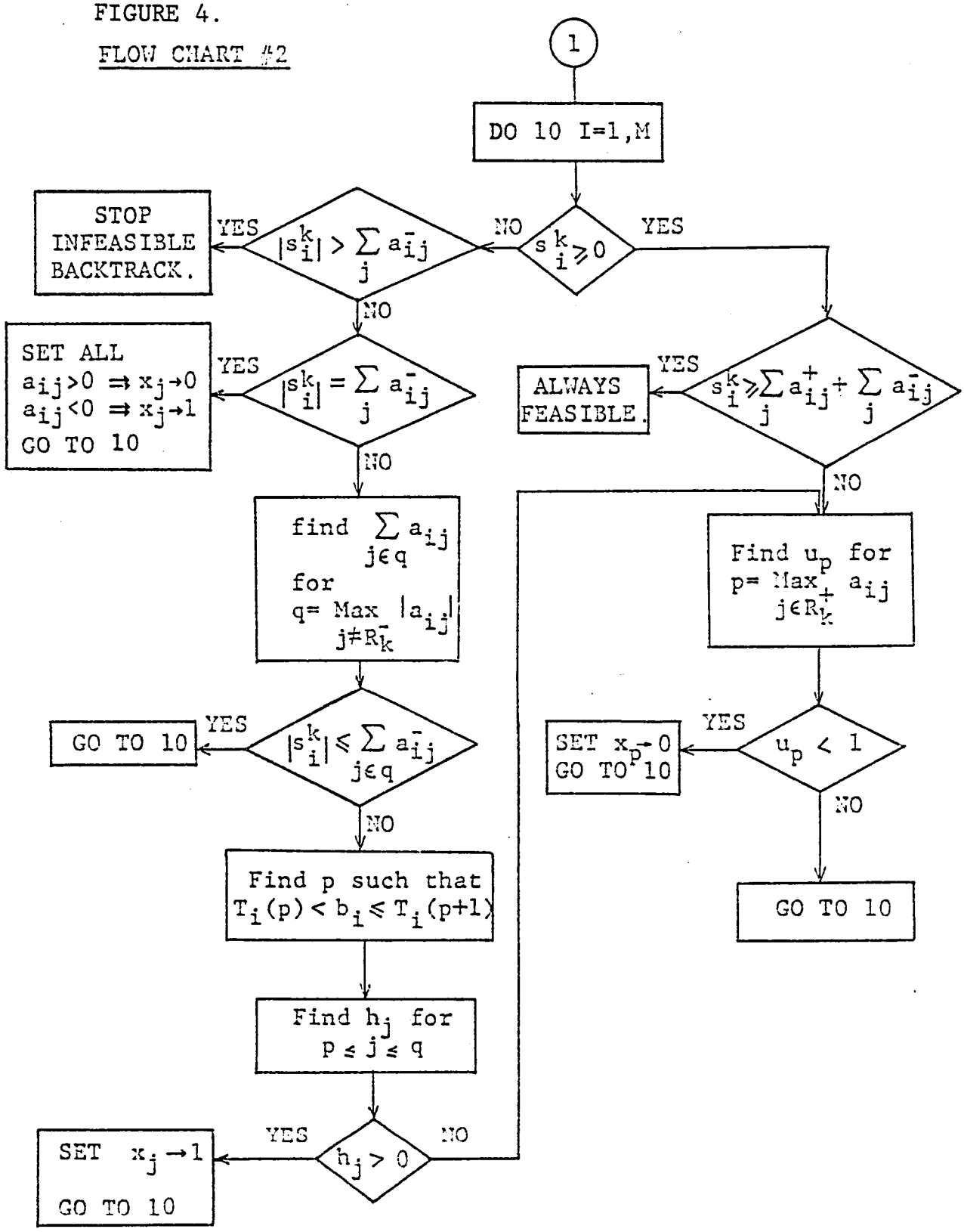


FIGURE 5.

FLOW CHART #3

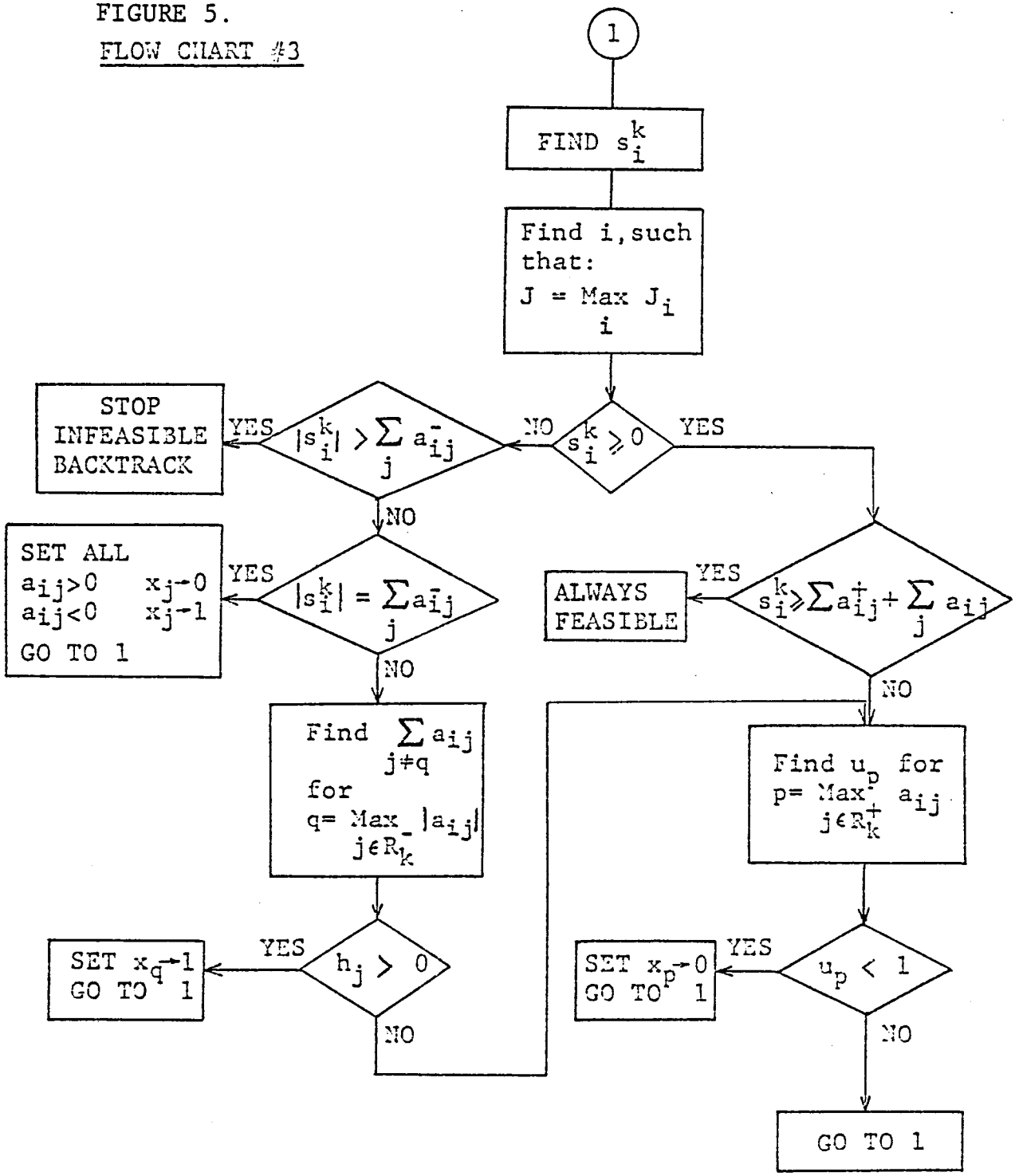
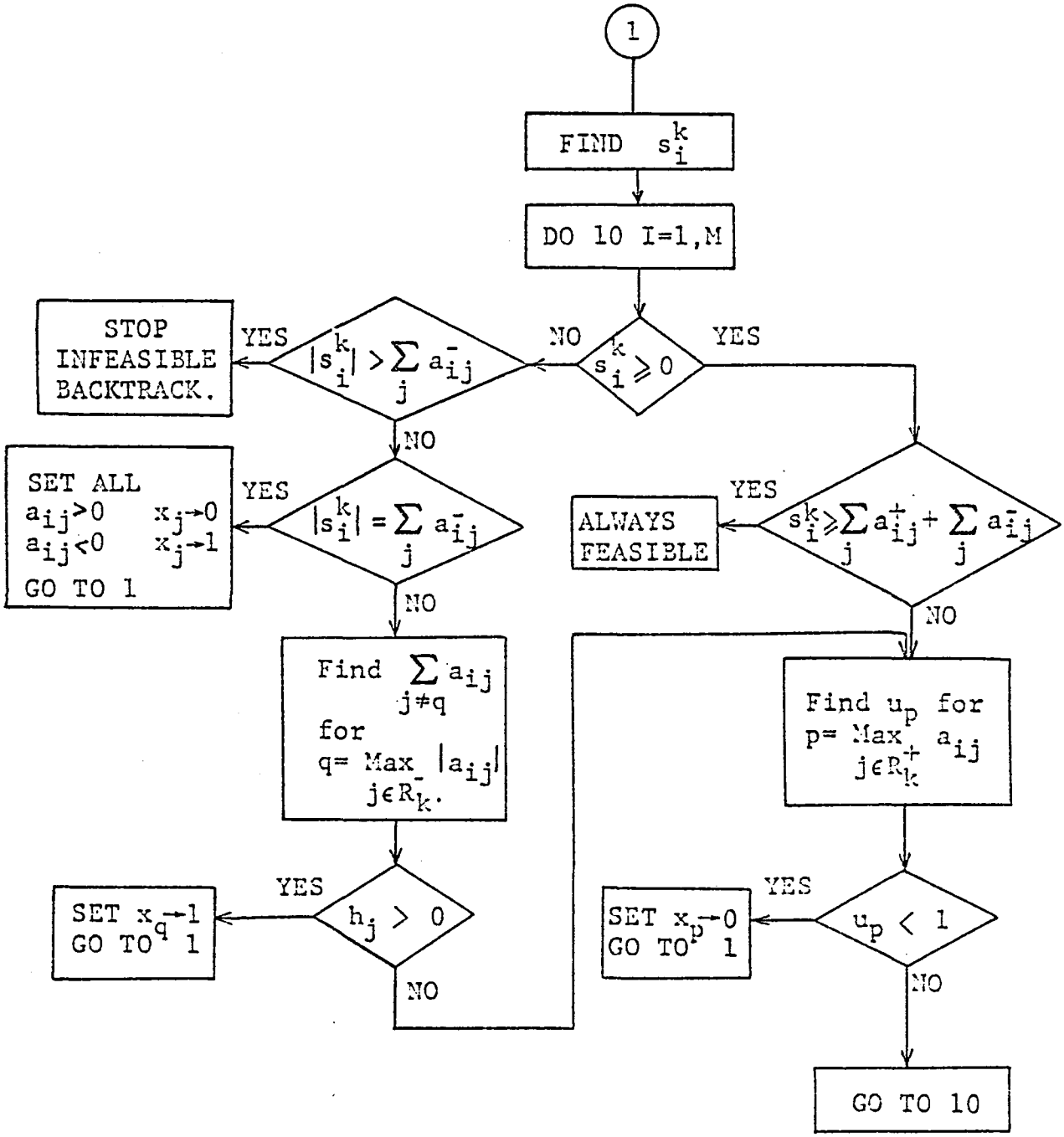


FIGURE 6.

FLOW CHART #4



For our purposes, we will distinguish between some very clearly defined approaches. We consider that in general, the goal of any branching rule most be to arrive sooner at an optimal solution if one exists. In particular, we will want to arrive as soon as possible at a point in which the different fathoming criteria already analyzed are more effective. One of these approaches can be based on the study of the objective function matrix, while another possibility is to consider the constraint matrix and the concepts of feasibility and infeasibility. A third approach will be obtained by taking into account the bounding on variables described in the last section.

BR1 THE OBJECTIVE FUNCTION APPROACH

One way to branch is looking at our goal of arriving as soon as possible at a point where the fathoming by BOUNDS will be effective.

Considering that the objective function matrix has in general, positive and negative coefficients, c_{ij}, we can try to see which branching rule will improve the fathoming by BOUNDS that we have already discussed in the preceding section.

As we have seen, we have two kinds of fathoming by bounds:

$$\text{F1} \qquad \bar{z}_k = \sum_{j \in S_k^+} c^j + \sum_{j \in F_{ki}^+} c^j < LB$$

$$\text{F2} \qquad \underline{z}_k = \sum_{j \in S_k^+} c^j + \sum_{j \in F_{ki}^-} c^j \ \theta \ UB(A)$$

Where θ means that at least one component has to be greater for every vector of UB(A).

Then, if we are at stage k of the problem and we are feasible and unbounded, we are in a stage in which the opposite relationship to F1 and F2 are satisfied. Now, we can try to see which variable to set and to what value, in order that each bounding procedure be more efficient at stage k + 1. It is clear that what we will need is that $\bar{z}_{k+1}$ be as low as possible and $\underline{z}_{k+1}$ as large as possible.

To achieve this we can analyze what variable we need to set to 0 or 1 for each of the two cases.

F1. At stage k, we have that

$$\bar{z}_k = \underbrace{\sum_{j \in S_k^+} c^j}_{K} + \underbrace{\sum_{j \in F_{ki}^+} c^j}_{F_k^+} \nleq LB$$

and at stage k+1, we would like that

$$\bar{z}_{k+1} = \underbrace{\sum_{j \in S_{k+1}^+} c^j}_{K'} + \underbrace{\sum_{j \in F_{k+1,i}^+} c^j}_{F_{k+1}^+} \leqslant LB$$

It is clear that in order to get $\bar{z}_{k+1}^+ \leqslant LB$, we need to decrease all components of $\bar{z}_{k+1}$ that are greater than its corresponding ones in LB. Let's study one of those components at stage k.

$$c_{ij} > 0 \begin{cases} x_j \to 0 & \left\{ \begin{array}{c} K' = K \\ F_{k+1}^+ < F_k^+ \end{array} \right\} \quad \bar{z}_{k+1} < \bar{z}_k \\ x_j \to 1 & \left\{ \begin{array}{c} K' = K + c_{ij} \\ F_{k+1}^+ < F_k^+ \end{array} \right\} \quad \bar{z}_{k+1} \simeq \bar{z}_k \end{cases}$$

$$c_{ij} < 0 \begin{cases} x_j \to 0 & \left\{ \begin{array}{c} K' = K \\ F_{k+1}^+ = F_k^+ \end{array} \right\} \quad \bar{z}_{k+1} = \bar{z}_k \\ x_j \to 1 & \left\{ \begin{array}{c} K' = K - c_{ij} \\ F_{k+1}^+ = F_k^+ \end{array} \right\} \quad \bar{z}_{k+1} < \bar{z}_k \end{cases}$$

As we can see in the last table, for an objective i and a variable j, if $c_{ij} > 0$ we set $x_j \rightarrow 0$, and if $c_{ij} < 0$ we set $x_j \rightarrow 1$, because what we need is $\bar{z}_{k+1} < \bar{z}_k$.

So far we know that a variable with a specific value has to be set to a determined value. But in general, for a particular variable x_j, we have $c_{ij} > 0$ and $c_{ij} < 0$ for different objective functions i and what we need is to decrease all the components of $\bar{z}_k$ that are greater than its corresponding one in LB. To decide what variable has to be set to one or zero, we need to define an overall coefficient for each variable j, at stage k, that gives us a measure of the sign of the coefficient of this variable, for all the objectives at the same time.

This overall coefficient, defined over all the objectives could be the following vector O(K) of dimension n-k and whose component j will correspond to a variable j at stage k, such that

$$O_j(k) = \sum_{i=1}^{p} c_{ij} \quad \text{for all } j \in F_k$$

Then, we can easily find which variable has to be chosen for branching. We choose a variable p such that

$$O_p(k) = \underset{j \in F_k}{\text{Max}} \; O_j(k)$$

and take $x_p \to 0$; if $O_p(k) > 0$.

We can find also another variable q such that

$$O_q(k) = \underset{j \in F_k}{\text{Min}} \; O_j(k)$$

and take $x_q \to 1$; if $O_p(k) < 0$.

F2. At stage k we have

$$\underline{z}_k = \underbrace{\sum_{j \in S_k^+} c^j}_{K} + \underbrace{\sum_{j \in F_{ki}^-} c^j}_{F_k^-} \not\geq UB(A)$$

and at stage k+1, we would like that

$$\underline{z}_{k+1} = \underbrace{\sum_{j \in S_{k+1}^+} c^j}_{K'} + \underbrace{\sum_{j \in F_{k+1,i}^-} c^j}_{F_{k+1}^-} \not\geq UB(A)$$

That is to say, at least one component of $\underline{z}_{k+1}$ is bigger than its corresponding in one of the vectors of UB(A). For all vectors of UB(A) and not necessarily the same component. Even though we only need one component, it is probably that not the same component will satisfy this requirement for every vector of UB(A).

It is clear then, that in order to get $\underline{z}_{k+1} \not\ni UB(A)$, what we need is $\underline{z}_{k+1} > \underline{z}_k$.

$$
\begin{array}{lll}
c_{ij} > 0 & x_j \to 0 & K' = K,\ F^-_{k+1} = F^-_k \quad \Rightarrow \quad \underline{z}_{k+1} = \underline{z}_k \\
 & x_j \to 1 & K' = K + c_{ij},\ F^-_{k+1} = F^-_k \quad \Rightarrow \quad \underline{z}_{k+1} > \underline{z}_k \\
c_{ij} < 0 & x_j \to 0 & K' = K,\ F^-_{k+1} > F^-_k \quad \Rightarrow \quad \underline{z}_{k+1} > \underline{z}_k \\
 & x_j \to 1 & K' = K - c_{ij},\ F^-_{k+1} > F^-_k \quad \Rightarrow \quad \underline{z}_{k+1} \simeq \underline{z}_k
\end{array}
$$

As we can see in this figure for an objective i and a variable j, if $c_{ij} > 0$, we set x_j at 1, and if $c_{ij} < 0$, at 0.

Therefore, we should not put together both types of fathoming by bounds because as we have seen, a branching rule that increases the potential of the fathoming is contradictory to both of them.

BR2 THE CONSTRAINT MATRIX APPROACH

Another way to branch is by taking into account the constraint matrix. Our objective in this case is to arrive as soon as possible at a point where the fathoming by feasibility is more effective.

$$t_i = \sum_{j \in F^-_{ki}} a_{ij} x_j > s_i$$

Considering that the constraint matrix has in general, positive and negative coefficients a_{ij}, we can try to determine which branching rule will improve the fathoming by feasibility that we have already discussed in the preceding section. It is clear, that if we are at stage k of the problem, we have a situation such that $t_i^k < s_i^k$. Then, what we need at stage k+1 is, if possible, that $t_i^{k+1} > s_i^{k+1}$. In order to

achieve this, we need $t_i^{k+1} > s_i^{k+1}$ or $s_i^{k+1} > s_i^k$.

As we can see in the next table, for a constraint i and a variable j, if $a_{ij} > 0$, we set x_j at 1, and if $a_{ij} < 0$, at 0.

			t_i	s_i
a_{ij}	0	$x_j \to 0$	$t_i' = t_i$	$s_i' = s_i$
		$x_j \to 1$	$t_i' = t_i$	$s_i' = s_i - a_{ij}$
a_{ij}	0	$x_j \to 0$	$t_i' > t_i$	$s_i' = s_i$
		$x_j \to 1$	$t_i' > t_i$	$s_i' = s_i + a_{ij}$

The problem in this case is to answer the following two questions:

-Do we have to consider overall coefficients or shall we look at individual rows?

-Do we need to look only at coefficients or shall we introduce the RHS's in our consideration?

We will discuss these questions after analyzing the following branching rule.

BR3 THE BOUNDING IN VARIABLES APPROACH

Another way to branch is by considering the bounding in variables, since we know that this feature decreases the computational time substantially.

As we have seen, there is a fundamental difference between constraints with RHS's positive and negative. If the RHS is negative, we can carry the problem to a point where it is infeasible, when

$$|b_i| > \sum_{k \neq j} a_{ik}^-$$

Besides, in this case, we can find two variables with a possibility of being bounded. One negative of the lower bound is greater than zero, and another positive with the upper bound less than one.

If the RHS is positive, in order to be able to say that the problem is infeasible we have to drive the RHS to a negative value. Besides, we have the possibility that for a constraint the problem is always feasible, when

$$b_i \geqslant \sum_{k \neq j} a_{ik}^+ + \sum_{k \neq j} a_{ik}^-$$

In this case, we can only find one variable with the possibility of being bounded, a positive variable when the upper bound is less than one.

This third approach is essentially similar to the second one, but with the particularity that now we can answer one of the questions previously raised. Since only one single infeasible row is enough to make the whole problem infeasible, we do not need to find overall coefficients. We will then examine only single rows of the constraint matrix and in particular those with negative RHS. In this case, if at stage k the problem is feasible, for a negative constraint we have:

$$\sum_{j} \bar{a}_{ij} > |s_i^k| \; : \; i \in Q_k^-, \quad j \in F_{ki}$$

and even more, if no variables have been bounded, we know that

$$\sum_{j \neq q} \bar{a}_{ij} > |s_i^k| \; : \; q = \underset{j \in R_k^-}{\text{Max}} \; a_{ij}$$

Our purpose is to drive the problem towards points at which the bounding on variables is more effective and, if it is possible towards infeasibility.

To accomplish this situation requires that at stage K+1

$$\sum_{\substack{j\neq q \\ j\in F_{ki}}} a_{ij}^{-} < \sum_{\substack{j\neq q \\ j\in F_{k+1,i}}} a_{ij}^{-} \quad \text{or} \quad |s_i^k| > |s_i^{k+1}|$$

As we can see in the next table, the solution is the same as in BR2, but now we have a very good idea of how to branch. If we have some constraints with negarive RHS's, we can choose: If $a_{ij} > 0$, set x_j at 1, and if $a_{ij} < 0$, at 0.

		for $i \in Q_k^-$ $\sum_j a_{ij}^-$	$\lvert s_i^k \rvert$
$c_{ij} > 0$	$x_j \to 0$	EQUAL	EQUAL
	$x_j \to 1$	EQUAL	↑
$c_{ij} < 0$	$x_j \to 0$	↓	EQUAL
	$x_j \to 1$	↓	↑

Instead, for constraints with positive RHS's, (in the event that we do not have any with a negative RHS), our goal

is to drive the constraint towards negative RHS's, in which case the best way to branch is: If $a_{ij} > 0$, set x_j at one.

Still, we need to know which row to choose. The procedure that we suggest is the following:

-If we have some rows with negative RHS's we will choose among them:

$$1 = \underset{i \in Q_k^-}{\text{Min}} \left(s_i^k + \sum_{j \neq q} a_{ij}^- \right); \quad q = \underset{j \in R_k^-}{\text{Max}} |a_{ij}|$$

For the case in which we do not have any negative RHS row, we will choose the following one:

$$1 = \underset{i \in Q_k^+}{\text{Min}} \left(s_i^k - \sum_{j \neq q} a_{ij}^- - \sum_{j \neq p} a_{ij}^+ \right); \quad p = \underset{j \in R_k^+}{\text{Max}} a_{ij}$$

BR4 THE FEASIBILITY APPROACH

Another method is the traditional branching towards feasibility. As Geoffrion suggests, branching towards feasibility has the advantage that we will arrive sooner at a possible efficient point and therefore we can increase our lower

bound set.

Branching towards feasibility requires the use of overall coefficients. Garfinkel and Nemhauser [18] suggest a way to find which variables to choose.

$$I_k = \sum_{i=1}^{m} \text{Max } (0,-s_i) = - \sum_{i \in Q_k^-} s_i$$

is the overall infeasibility at stage k. We can find the infeasibility that at stage k+1, the branching in variable j will produce:

$$I_k(j) = \sum_{i=1}^{m} \text{Max } (0,-s_i+a_{ij})$$

and then we will choose variable l such that

$$I_k(l) = \min_{j \in R_k^-} I_k(j)$$

BR5

Finally, we can use a combination of branching rules as suggested in Breu and Burdet [11].

A combination of the BR1 and BR4 criteria will be

$$\lambda I_p(k) + (1-\lambda) O_p(k)$$

where for $\lambda=1$ is absolute feasibility branching and for $\lambda=0$ is absolute objective branching.

3.2.5 Dominance Test

Testing dominance is not so simple if we want to do it in the fastest possible way. We have already seen in Villarreal's work that the dominance test is one of the most time consuming.

In order to better develop the explanation of the different ways to perform this test we distinguish two different kinds of dominance tests:

-DIRECT DOMINANCE. Once we have found an end point in the implicit enumeration algorithm (we will call this point "feasible", that is to say that it is not fathomed either by the feasibility test or by bounding) we test if this point is dominated by any of the vectors already on the list.

-INVERSE DOMINANCE. On the other hand, we can also test if this point which we have found, dominates any of the points already on the list.

Having clarified this concept, we will analyze the different strategies.

DT1

One of the most complete dominance tests that we can perform is the following: We begin by determining for each feasible point found, the variable SC, the sum of its p components.

If EF(i,j) is the matrix of all the points that are included in the list,

$$SC(j) = \sum_{i=1}^{p} EF(i,j) \quad \text{for each } j.$$

Since we know that if we have two feasibles points k,l, and SC(k) > SC(l), the vector l can not dominate vector k, we can proceed as follows:

We form the matrix EF(i,j) by ordering all vectors j from

the largest to the smallest in terms of SC. Then, each time we find a new feasible point m, we determine its SC(m) and we put it in its correct place (m) in the matrix. Then, we only have to test dominance in the following way:

-We test direct dominance from the first vector in the list (greatest SC) until the vector in the place m-1, (which are the only ones that can dominate vector m). It is understood that if one of these dominates point m, we do not need to continue the test, and vector m may be deleted.

-If point m in not eliminated, then we test inverse dominance from the vector in place until the end. These are the only ones that can be dominated by vector m. We delete all points dominated by m.

The problem at this point is to know if the time gained with the procedure compensates the time spent in ordering matrix EF.

DT2

The simplest way to perform this test and find all the efficient points of the problem is simply to introduce all the feasible points in the matrix EF and at the end of the

algorithm perform direct and inverse dominance, or use the algorithm in Bitran [10] .

We begin by comparing the first element of EF with each vector on the list, performing first direct dominance and afterwards inverse dominance each time. All the points dominated by the first vector are deleted without testing inverse dominance, and in the moment that a point dominates the first element it is automatically put in the place of this first vector and the procedure is restarted.

If the first element is not inversely dominated by any other on the list, we know that it is efficient and we continue the procedure by going to the second element.

Although this method is very simple, it is not necessarily inferior to the others, because where we find a good efficient point, we will eliminate a lot of feasible solutions in the list by direct dominance.

Another feature of this algorithm is that when in the comparison of two solutions, if a component of the first vec-

tor is greater than its corresponding one in the second vector and another component of the second vector is greater than its corresponding one in the first vector, we do not continue making more comparisons.

The drawback of this procedure is the dimension of EF, since all the feasible points of the problem are included on the list. Then, the capacity of the computer will be the cut off point of the method.

DT 3

Another way to perform the dominance test, and a good compromise between the last two procedures is the following:

Each time we find a feasible point we perform direct dominance. Then, if the feasible solution is dominated by any of the points already on the list it is deleted. Otherwise, we include it in the list without further consideration.

At the end of the algorithm we perform dominance to find all the efficient points.

Since we have previously found some lower bounds that, as we will see, are good efficient points, this procedure can

perform well and has the advantage of a low dimension matrix EF.

3.2 A Revised Enumeration Algorithm

Having studied the different parts of an implicit enumeration algorithm we can now try to construct it. We present below a rough scheme of what we need to do (Figure 7):

(I) Find efficient points over unit cube (A).
EF(A) = UB(A) U LB(A). Those that are feasible in our problem we will call LB(A) and the others UB(A).

(II) INITIALIZATION. At v_o, we know that $\underline{z}_o$ = LB(A) and $\overline{z}_o = \infty$ and we want to find EF(P) that we know can be written as: EF(P) = LB(A) U RD, where RD has to dominated by UB(A). Go to s p (III).

(III) BOUNDS. At v_k we find $\overline{z}_k$ as follows:

$$\overline{z}_k = z_k + UB(F) = \sum_{j \in S_k^+} c^j + \sum_{j \in F_k^+} c^j$$

Note that if we find an efficient point of our original problem (P), we can form $\underline{z}_k$= LB(A) U VEF(P).

(IV) FATHOMING. If for any constraint i

A) $t_i > s_i$

B) $\bar{z}_k \leqslant \underline{z}_k$

C) $\underline{z}_k \geqslant UB(A)$

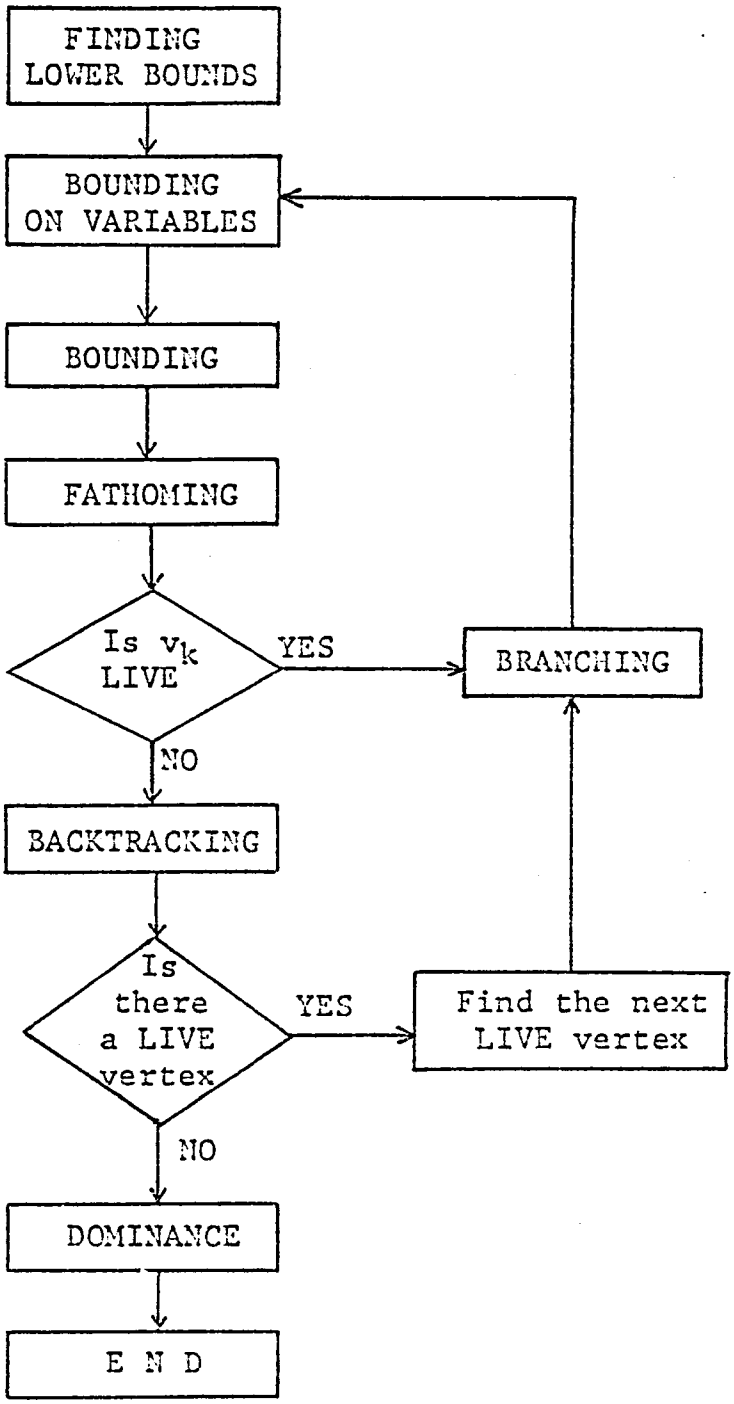
v_k is fathomed and we go to step V. If v_k is live go to step VI.

(V) BACKTRACKING. If no live vertex exists, go to step VII, otherwise branch to the newest live vertex and go to step III.

(VI) PARTITIONING AND BRANCHING.

(VII) TERMINATION. We have found a set of points among which we have the set of efficient points. By applying the definition of efficiency among them, we will obtain EF(P) = LB(A) U RD.

FIGURE 7.


FINDING LOWER BOUNDS
BOUNDING ON VARIABLES
BOUNDING
FATHOMING
Is v_k LIVE
YES
BRANCHING
NO
BACKTRACKING
Is there a LIVE vertex
YES
Find the next LIVE vertex
NO
DOMINANCE
E N D

SECTION 4

THE ALGORITHM

4.1 The Program

The algorithm coded in Fortran IV is based on a Simple Implicit Enumeration Algorithm with some modifications that we have already discussed in the last chapter.

The first algorithm coded is the one described in section 4.2. Afterwards, several improvements in finding lower bounds and in performing dominance tests were made. A second algorithm is obtained and computational results are reported.

The procedure is as follows: At each vector or node we choose a variable to be fixed and we branch on it. We continue the procedure until the point is fathomed at some stage of the problem, by feasibility (constraints or bounding), or we arrive at an end point, where we test dominance.

We therefore have three types of nodes: The Root Node,

Intermediate Nodes, and End Nodes. Now we need to characterize each type of node.

ROOT NODE.	FEASIBLE:	BRANCH
	INFEASIBLE:	STOP. (INEFFICIENT)
INTERMEDIATE NODES.	FEASIBLE:	BRANCH
	INFEASIBLE:	BACKTRACK. (DISCARD)
END NODES.	FEASIBLE:	BACKTRACK. (ADD POINT)
	INFEASIBLE:	BACKTRACK. (DISCARD)
	DOMINATED:	BACKTRACK. (DISCARD)

So, we need a variable to differentiate each kind of node. To do this we create variable LEV (level in the tree). For a problem with n components:

$$\text{LEV (ROOT)} = 0$$

$$0 < \text{LEV (INTERMEDIATE NODE)} < n$$

$$\text{LEV (END NODE)} = n$$

It is clear that for branching we only have to do LEV=LEV+1 and for backtracking LEV=LEV-1.

Once that we know how to distinguish between nodes we need to know which variables have been set and to which value. To do this, we need two more variables:

VAR(i). Only LEV entries are significant.

VAL(i). Only LEV entries are significant.

VAR indicates the variable that has been set to any specific value at each step. VAR is then a vector of components i=1,...,LEV.

VAL indicates the value to which every selected variable has been set. It is also a vector of components i=1,...,LEV. Both vectors work together and inform us, which variable has been set and to what value, at each step of the problem. We have to UPDATE VAR and VAL when we BACKTRACK and BRANCH.

Still, we have to determine what values a variable can take. Although only two values are possible, zero and one, it is necessary to differentiate between variables set by branching or by bounding on variables. Because, in this last case the variable is set to zero or one definitely and when

we backtrack we will pass over them.

Thus, the value that a variable i can take at stage k is:

VAR(k) = i

VAL(k) = 0. x_i has value 0 by BRANCHING

VAL(k) = 1. x_i has value 1 by BRANCHING

VAL(k) = 2. x_i has value 0 by BOUNDING

VAL(k) = 3. x_i has value 1 by BOUNDING

This solves the problem of recognizing a variable when we backtrack. Backtracking will go backward in the branch until we find a variable whose value is zero or one. If the variable has a value of two or three we pass over it.

When we arrive at a variable set to zero or one we will take its opposite value but without repeating the same variable. We proceed as follows: If at stage k, variable x_i has VAL(k)=0, when we complete the backtracking we set x_i to one, and put VAL(k)=3. If variable x_i has VAL(k)=1, when we complete the backtracking we set x_i to zero and we put VAL(k)=2.

Branching chooses a variable not among the first LEV variables in VAR according to rules that we will discuss later.

In order to know if a variable is free or fixed, we have to look over the whole vector VAR(k). In order to avoid this, we keep another n-vector S(j). For each variable j, S(j) has two values: zero if x_j is free and one if it is fixed.

Finally, we need to know at each step of the algorithm, the value of the objective functions as well as the values of the RHS's of the constraint matrix. To achieve this, we use two more n-vectors: Z(i) representing the sum of the coefficients of objective i (i=1,...,p) for all variables fixed to one ($j \in S^+_{ki}$) and B(i) denoting the RHS vector of the constraint i (i=1,...,m).

Therefore, we need to keep track of Z(i) and B(i) throughout the problem. This raises the necessity of updating not only LEV, VAR, VAL, and S, but also Z and B, each time we branch on certain variables or as we shall see when we backtrack in the algorithm.

4.2 The First Algorithm

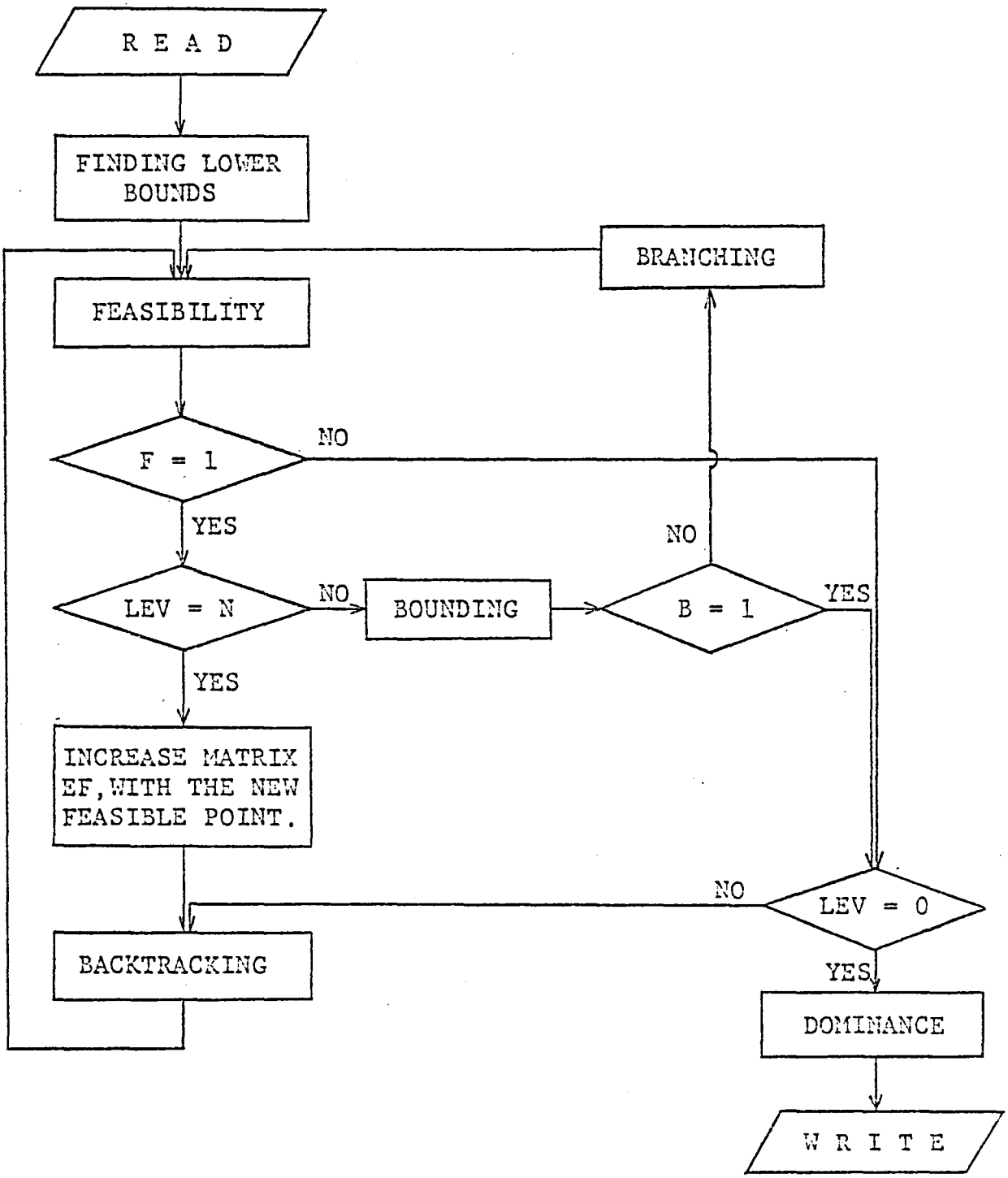
As we can see in the block diagram shown below we have a series of subroutines such as finding lower bounds, feasibility, backtracking, branching and dominance which we shall now explain (Figure 8):

The matrix EF is the matrix of feasible points which at the beginning of the algorithm represents the set of lower bounds and at the end will give us the set of efficient points. When we arrive at an end branch (LEV=n) still feasible, we add the point to this matrix EF. This is represented by EF = EF U Z, where Z, as we have seen, is the vector of the values of each objective.

4.2.1 Finding Lower Bounds

In this first algorithm we have coded a simpler version of the unit hypercube method of finding lower bounds discussed in Chapter III. This approach performed well in the initial stages of development. Later, when we began to use objective functions with all positive coefficients, the only lower bound

FIGURE 8. FIRST ALGORITHM



found was the vector of one's which, on the other hand, was usually infeasible.

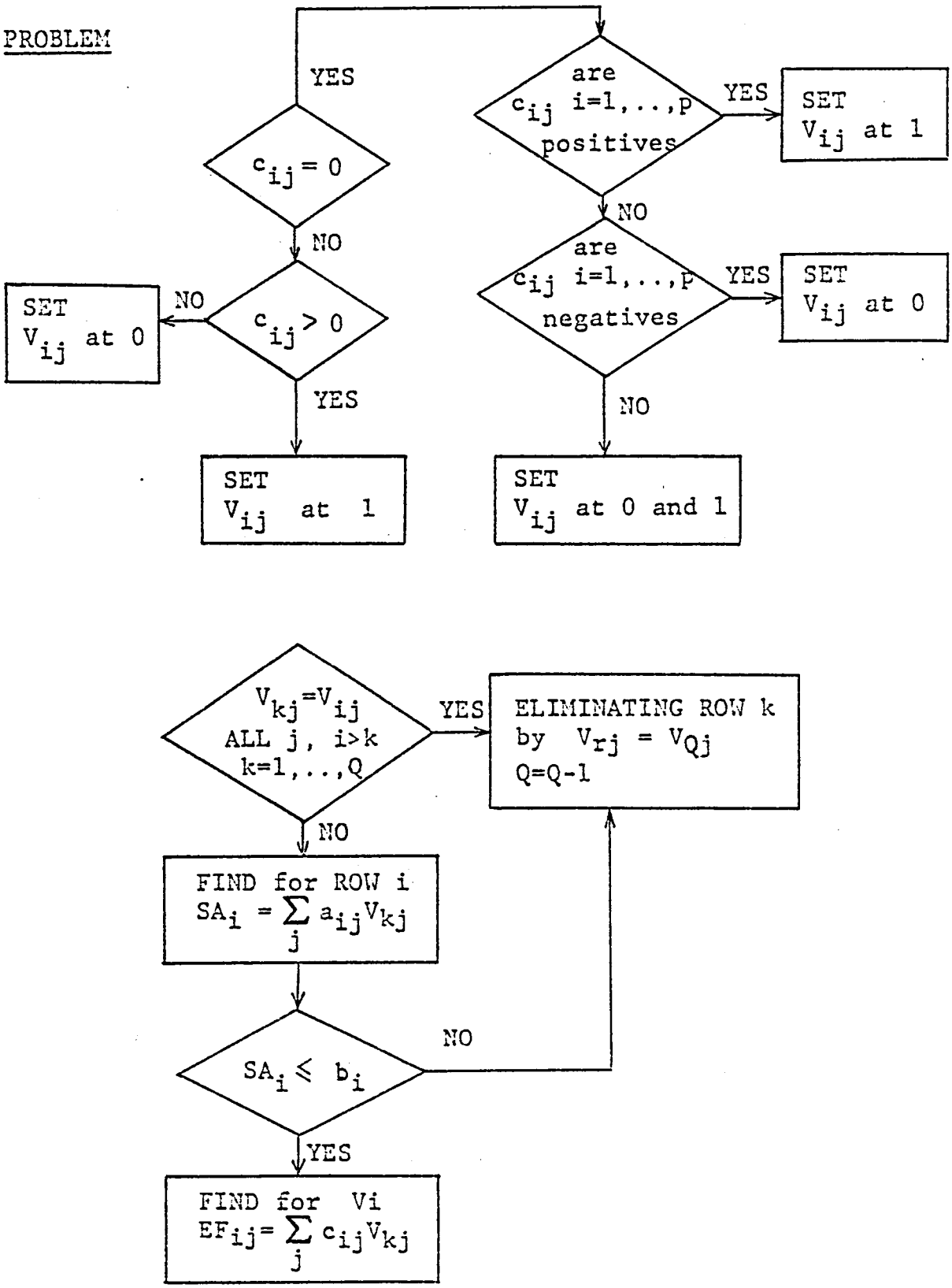
Although the procedure already discussed could give us all the efficient points of the auxiliary problem, we opted for coding a less complicated and undoubtedly less time consuming approach.

As we show in the following block diagram, we only find some of the efficient points of the unit hypercube, the ones most easy to find, by maximizing each one of the objective functions. This way we do not have to test dominance and we only test feasibility. If the point is feasible, we keep it in the matrix EF of future efficient points, and if not, we disregard it (Figure 9).

This subroutine begins by finding the matrix V, according to the sign of the coefficients c_{ij} of the objective functions (we do not consider in this first approach $c_{ij}=0$). We take $v_{ij}=1$ if $c_{ij} > 0$ and $v_{ij}=0$ if $c_{ij} < 0$. Afterwards, we eliminate all the rows of V that have identical components. Finally, we test feasibility and from the vectors that are

FIGURE 9.

THE AUXILIARY PROBLEM



feasible, we find the value of the objective functions which will initiate the matrix of efficient points EF.

The procedure of eliminating a row of the matrix is to simply transfer the last point of the matrix to this row and reduce the matrix number of rows by one.

4.2.2 Feasibility

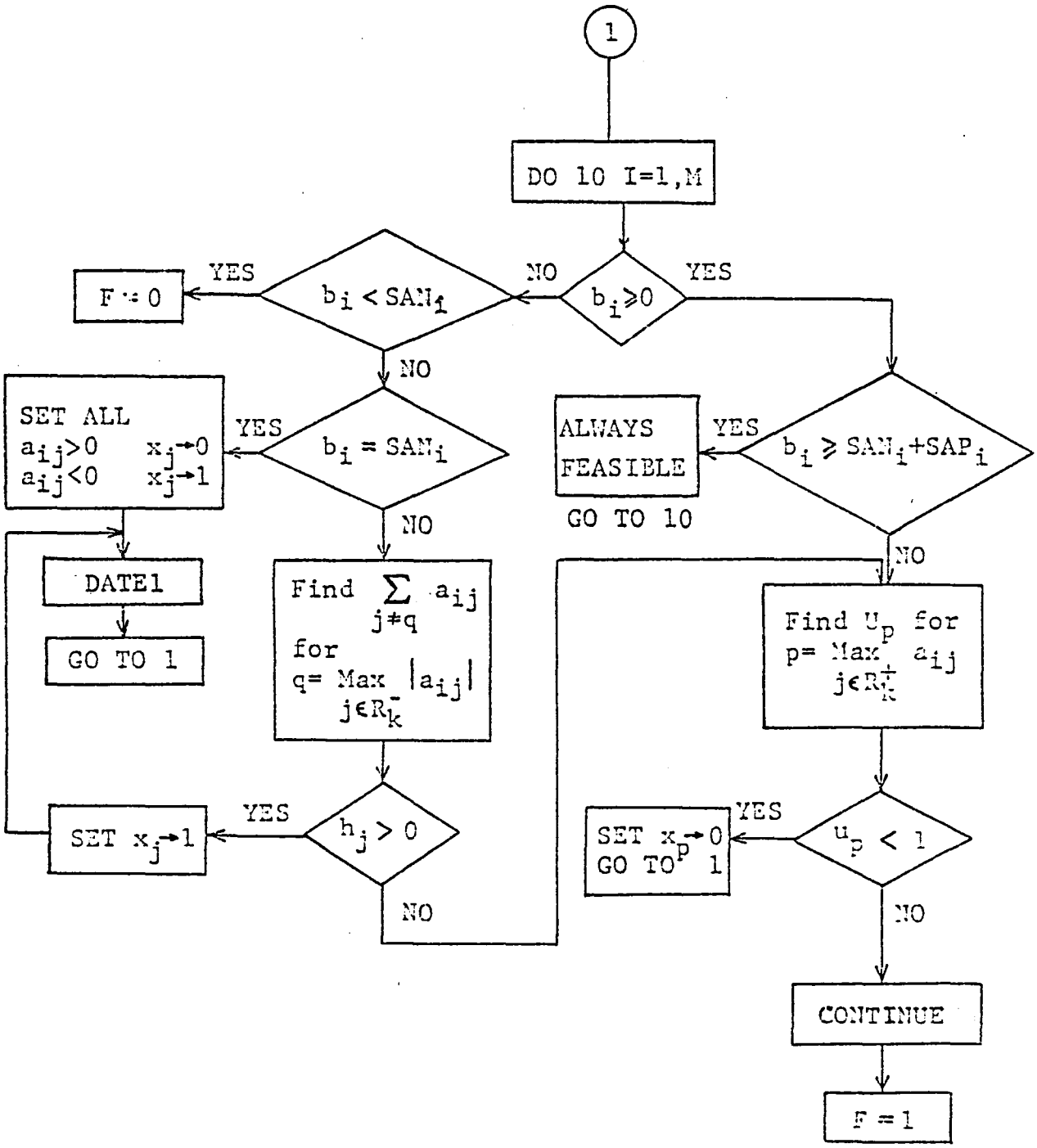
Among the four procedures discussed for this subroutine on page 86-96, we decide to code the number four.

The next block diagram demonstrates how the set of constraints is divided into two categories, depending upon the sign of its RHS. If the RHS is negative we test feasibility and find the lower bound for the most negative variable and the upper bound for the most positive one. If the RHS is positive, we test to see if the constraint set is feasible and also find the upper bound for the most positive variable (Fig. 10).

If, in any of the constraints, the sum of its negative coefficients (SAN) is greater than the RHS (for $b(i) < 0$) the

FIGURE 10.

FEASIBILITY



problem is infeasible (binary variable F=0). If all the constraints are feasible, the problem is feasible at this stage (F=1).

Note, that when we have LEV=n, (end of a branch), the procedure will perform in the same way. In this case, there will not be any LHS, therefore SAN = SAP = 0. (SAP is the sum of all positive coefficients of a given row constraint). Thus, if any RHS is negative the problem is infeasible, and if all RHS's are positive the problem is feasible.

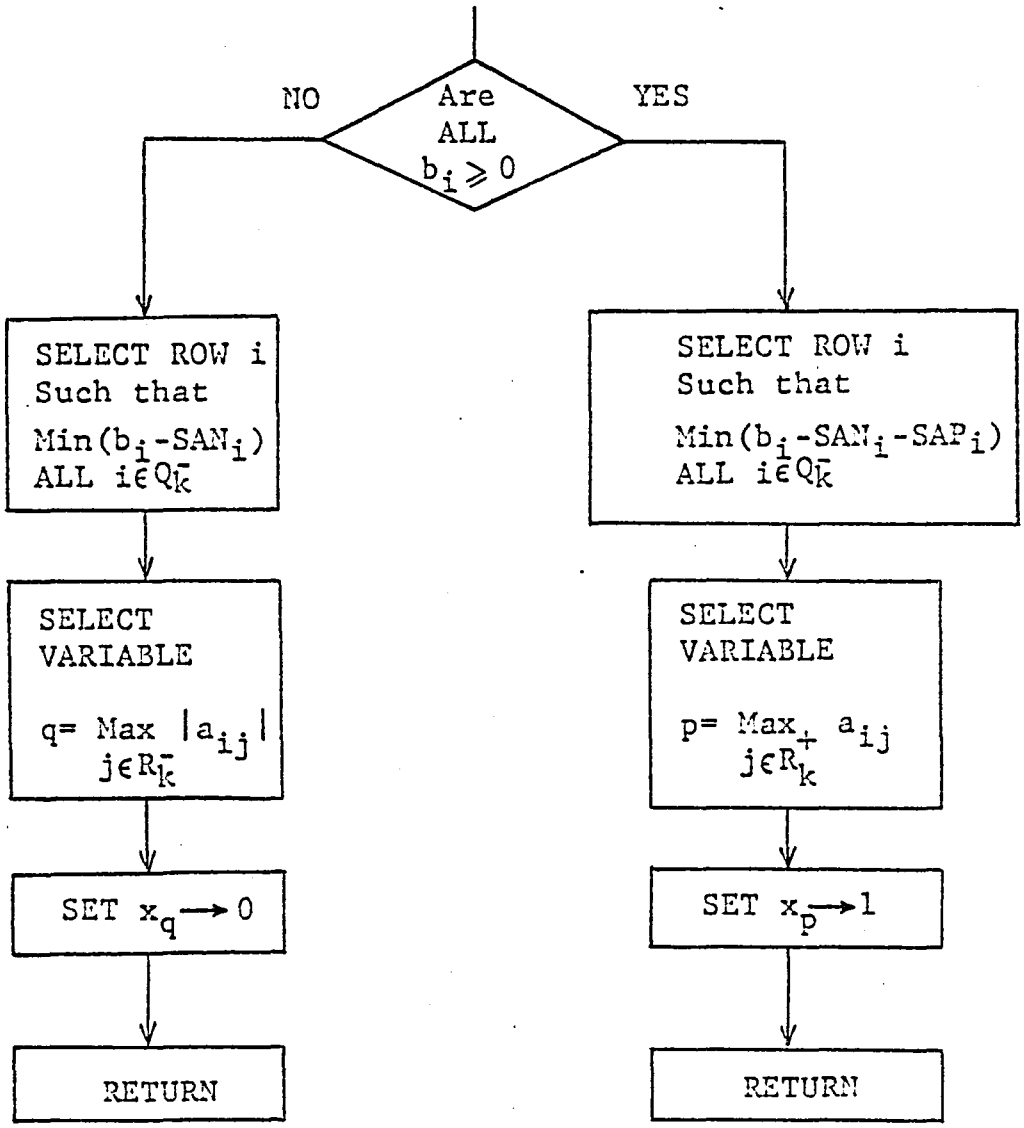
4.2.3 Branching

The next flow chart shows how we apply the third branching rule (BR3), discussed in the last chapter (Figure 11).

We use a binary variable IS which tells us if all the RHS's are positive (IS=1) or if at least one RHS is negative (IS=0). Since we finished subroutine feasibility ending up with the sum of the negative elements SAN(i) for each row i, as well as the sum of the positive elements SAP(i), we do not have to find them again. In order to know if a variable, j,

FIGURE 11.

BRANCHING



is fixed or free we use the end vector S(j) previously defined.

One essential point in the last two subroutines is the problem of updating the algorithm each time we set a variable to zero or one. If we set a variable to zero we do not have to update anything because neither the RHS, B(i), nor the solution of the problem at stage k, Z(i) will change.

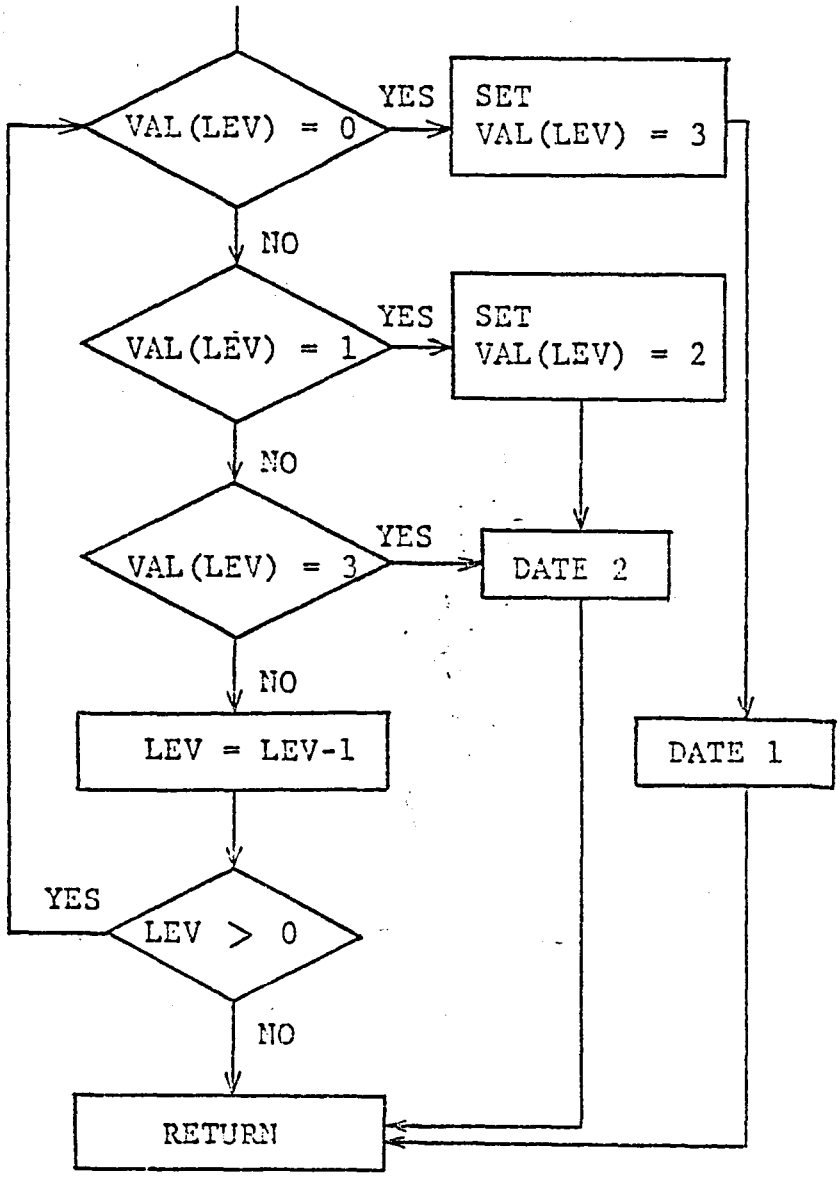
The problem arises when we set a variable to one. In this case, we have to change the RHS's of the constraints by the amount of the coefficient of the respective fixed variable and at the same time find the new vector value Z(i) of the solution of the problem at stage k+1. This kind of updating subrouting is called in the program DATE1, in order to differentiate it from the updating problem that will appear in backtracking.

4.2.4 Backtracking

The backtracking subroutine, as its name indicates, consists of going backwards in the branch of the tree until we

FIGURE 12.

BACKTRACKING



find a variable in which it is feasible to branch. Thus, we backtrack until we find a variable that has taken one of its two possible values (0 or 1 by branching), and automatically we set it to its opposite value (1 or 0) (Figure 12).

One essential point in the backtracking is the updating problem that appears when we go through a variable which either has taken its two possible values or it's set by bounding to a specific value. Note, that those variables are easily recognized because their current value is 2 or 3 as we pointed out on page 116.

If the value is 2, which means 0 by bounding, we do not have to update anything. If the value is 3, which means 1, we have to update exactly in the opposite way as we did in branching. This updating procedure is called in the program DATE2.

When we branch we have to update the problem again as we did in branching. If the value is 0 we will set the variable to 3 in order to show that it has taken its two possible values, and update with DATE1. If the value is 1 we will set

the variable to 2 and update with DATE2. This way, when we backtrack we will pass over the variables that have taken their two possible values and we will not repeat them in the branching.

4.2.5 Bounding

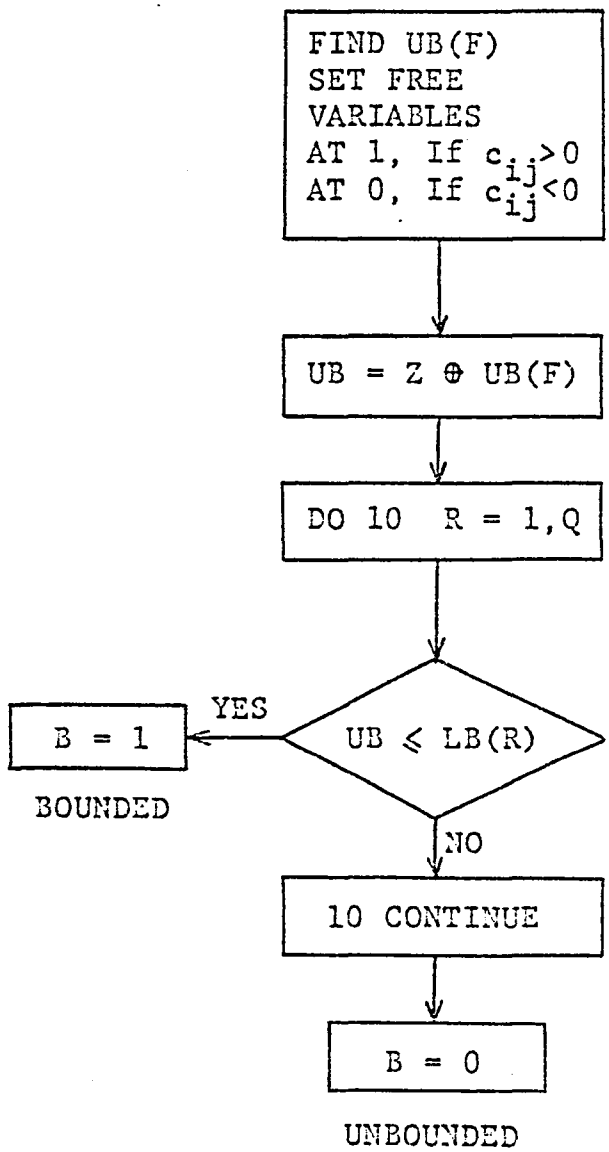
At a given stage of the problem we will have a set of lower bounds which is located in the matrix EF. They are not only the lower bounds found at the beginning of the algorithm, but also every feasible point currently found in the procedure.

At the same stage, we have the vector value Z(i) (i=1,...,p) of the solution of the problem at stage k. Then, we only have to look for the upper bound of the free variables UB(F). The UB(F) that we coded here is found by setting each free variable to 1 if $c_{ij} > 0$ or to 0 if $c_{ij} < 0$. The upper bound of the whole problem UB will be $UB = Z \oplus UB(F)$ (Figure 13).

Thus, the problem will be bounded (B=1) if all components of UB are less than or equal to any of the lower bounds of EF.

FIGURE 13.

BOUNDING



Again, a variable j is easily recognized as free if S(j) is equal to zero.

In this first algorithm, the bounding test is performed from the beginning of the problem (LEV=0), with the only condition that there is at least one lower bound in the matrix EF (Q≠0), where Q is the number of rows of this matrix at a given stage.

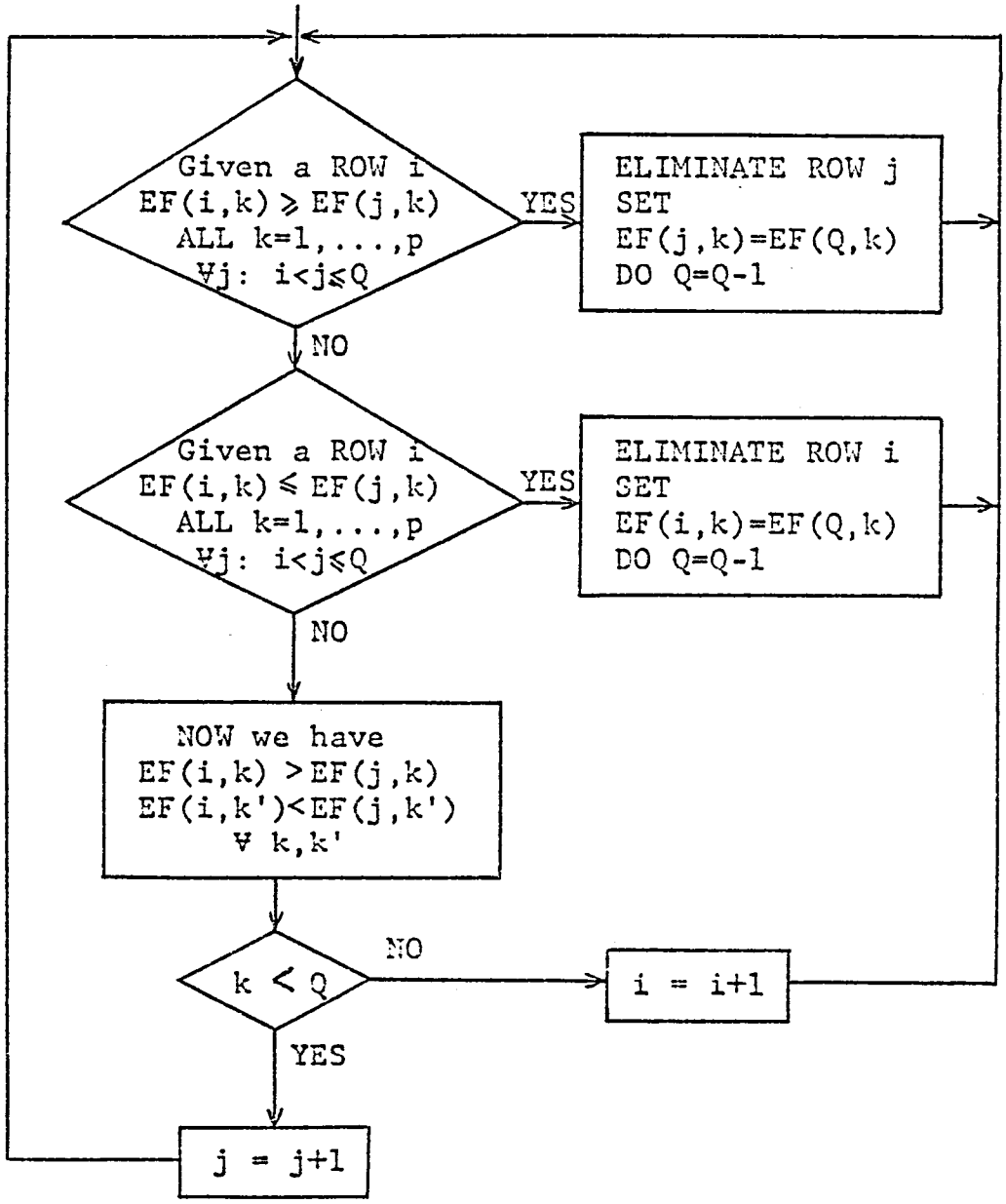
4.2.6 Dominance Test

The dominance test (direct and inverse) is performed at the end of the algorithm and for all feasible points of the problem. It is evident that if Q is equal to zero, there are no efficient points. If Q > 1, we perform dominance among the row vectors (feasible points of the problem) in order to find the set of efficient points (Figure 14).

We compare each row vector EF(i,h) h=1,...p; i=1,...,Q-1, with all the others EF(j,h), h=1,...,p; j=i+1,...,Q, eliminating dominated vectors by the procedure indicated in the next flow chart.

FIGURE 14.

DOMINANCE



Note that once we have compared row vector i with all j's, this vector i and subsequently all the others before i, are already efficient.

If we find that a component of a row vector i, is dominated while another dominates its corresponding one on a row vector j, we do not continue comparing these two vectors.

4.3 The Second Algorithm

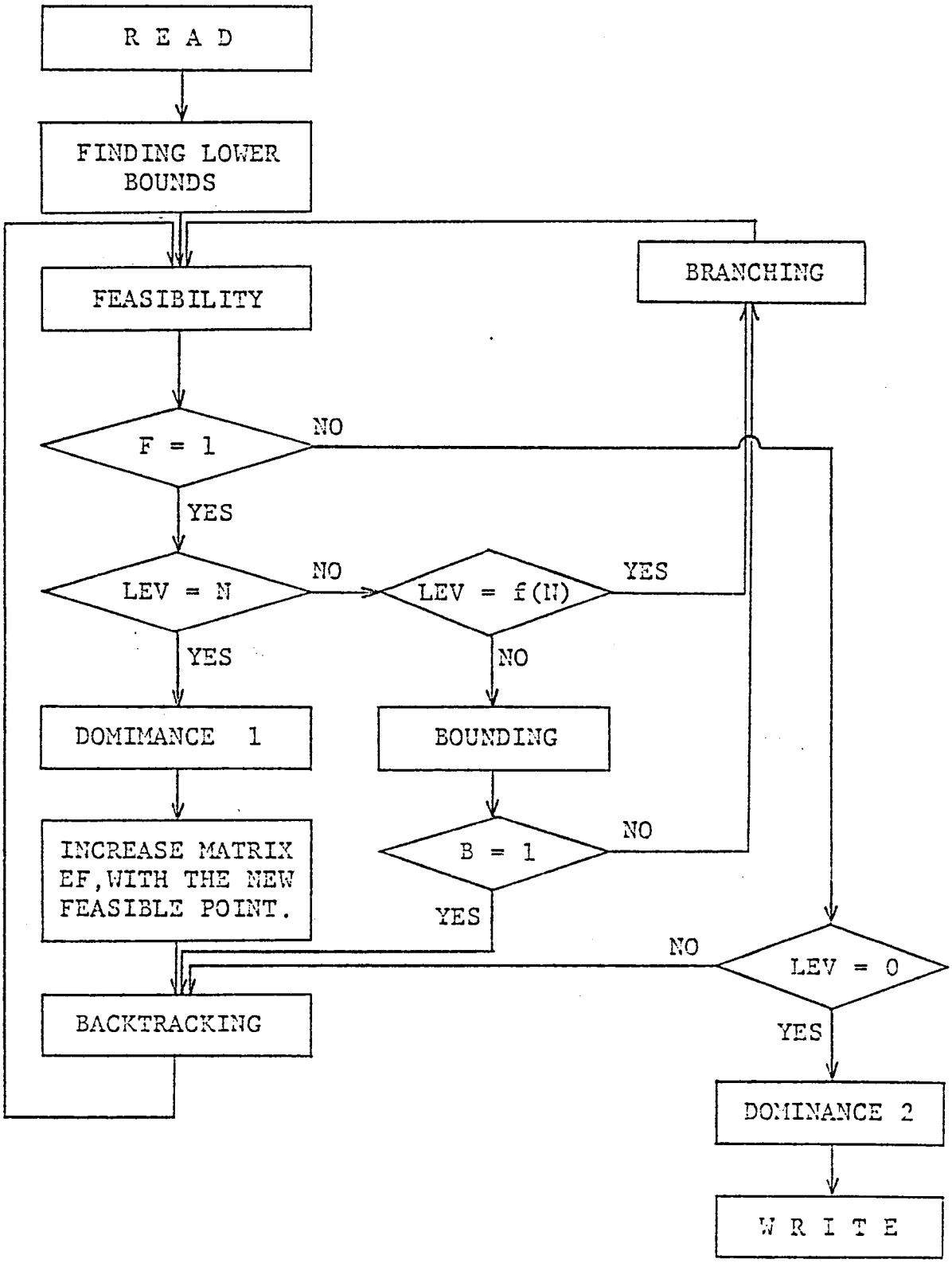
The improvements of this second algorithm over the first one are shown below. A new block diagram is also given on the next page (Figure 15).

The first problem that arises in the algorithm just described is the method of finding good lower bounds. Since we were using objective functions with all its elements positive, we had to use a method other than the auxiliary problem, and therefore we coded the λ's method of finding lower bounds.

We observed in the output of the first algorithm that

FIGURE 15.

SECOND ALGORITHM



the bounding procedure was not working very efficiently. Since it was clear that the test was not bounding at the earlier stages of the problem, we began bounding after a certain number of levels. We reached the $(n-5)^{th}$ variable stage, without experimenting any change.

At the same time we saw that we had obtained a very good lower bound by the λ's method. This lower bound not only was efficient but also very good for bounding (all components of approximately the same magnitude). Hence, the problem was that the upper bounds of the free variables were very high.

Another drawback of the first algorithm was that we often exceeded the capacity of the computer, generally when we increased the size of the problem to eighteen or twenty variables. In this case, the number of feasible points increased considerably and since we were performing dominance only at the end of the algorithm, the matrix EF of possible efficient points was getting bigger and bigger. In order to avoid this problem, we changed the dominance subroutine to the one described as DT3 in the last chapter.

4.3.1 Finding an Initial Lower Bound

In this new algorithm we used the λ's method previously discussed. With different combinations of λ's we could find different lower bounds.

For our problem of three objective functions we tried to find lower bounds over the whole set of possible efficient points. Then we used the following vectors of λ's:

$$\lambda_1 = (1,1,1)$$

$$\lambda_2 = (3,1,1)$$

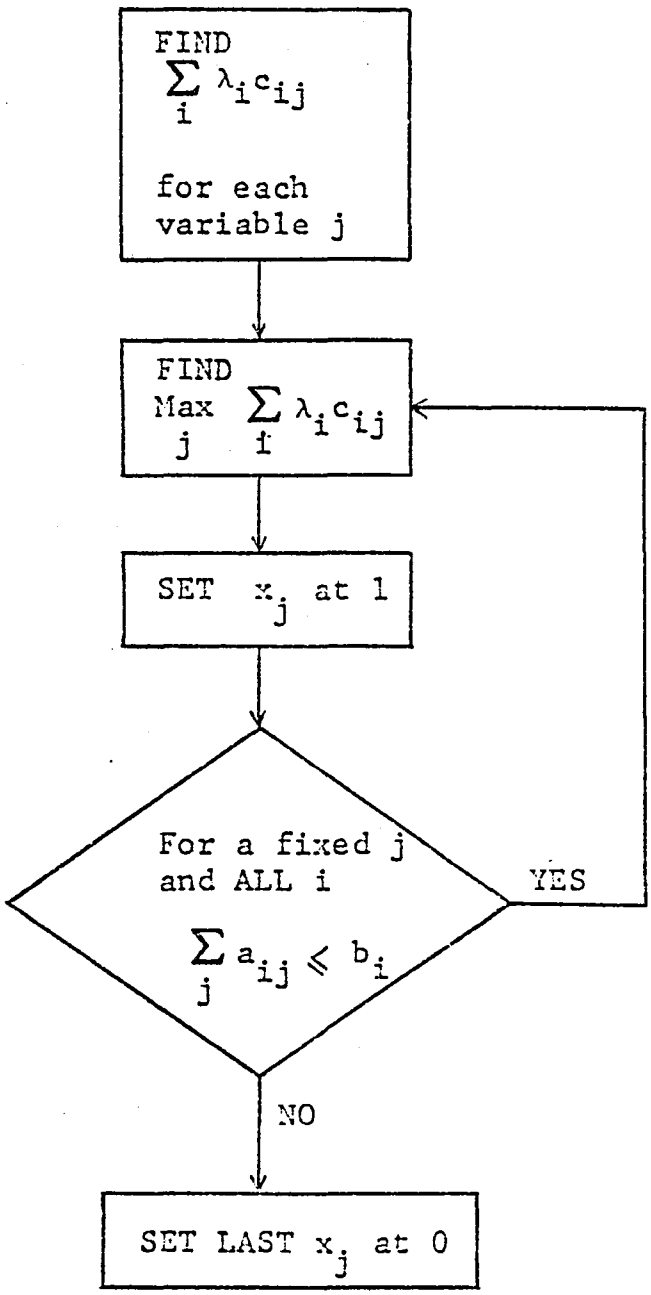
$$\lambda_3 = (1,1,3)$$

Each one of these λ_k will give us one lower bound which hopefully will also be an efficient point.

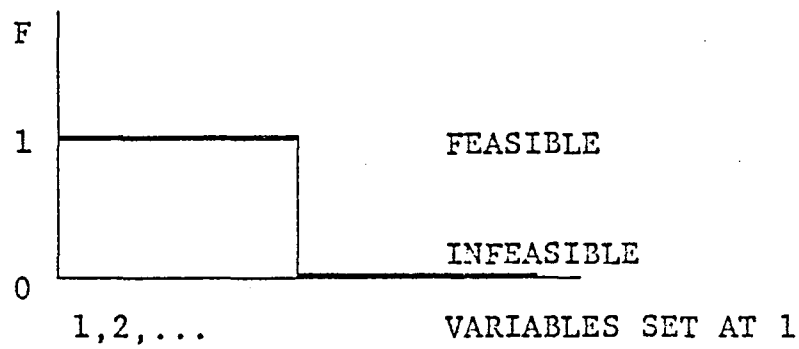
The procedure is described in the following block diagram. We create a new row from the objective functions, $\sum_i \lambda_{kj} c_i$, for each k=1,2,... Since for all variables equal to zero, the problem is feasible if all coefficients of the

FIGURE 16.

LOWER BOUNDS

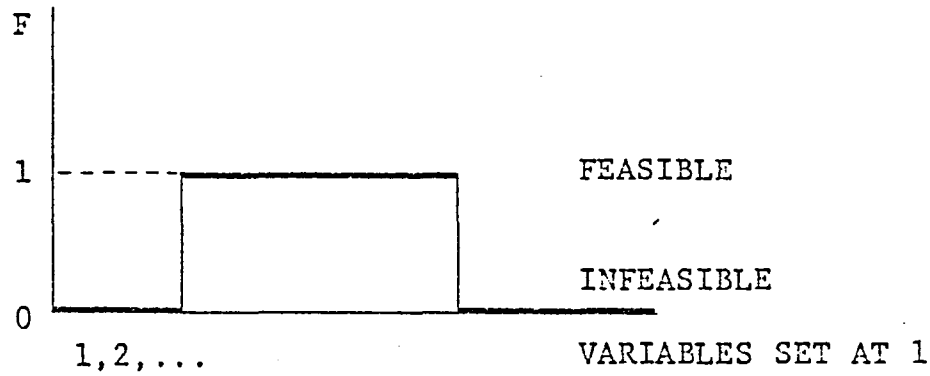


constraint matrix and the RHS's are positive, we begin by taking some variables equal to one, one at a time, until we reach infeasibility. In this case the last solution is the best that can be obtained (Figure 16).



Note that it is possible the feasibility region could be empty, in which case we cannot find the lower bound.

In general the procedure is not so simple. If we have RHS's positives and negatives, the solution with all variables set to zero is normally infeasible. Afterwards, the problem will become feasible when we have set enough negative variables to one. Finally the problem will be infeasible again when some positive RHS constraint is not satisfied. In this case, the last solution is the best.



4.3.2 Dominance Test

Once we determine an efficient point (LEV=n and feasible) we add it to the list of lower bounds EF. If Q, the number of vector rows of this matrix is greater than one, we test to see if this new vector is dominated by any of the feasible points already on the list; and if so, we eliminate it by simply doing Q=Q-1. If the feasible solution is not dominated, we continue the algorithm even knowing that this vector could itself dominate some other vector in the matrix.

The advantage is that since we find good efficient points by the lower bound method, this procedure will in fact be very effective.

Still, at the end of the algorithm, we again have to perform dominance, in order to find all the efficient points. Here, we use a slightly different approach than in the first algorithm. We use a binary variable IL which indicates to us if a component of the row vector i is dominated by its corresponding of j (IL=1), which allows us, to continue the procedure without comparing any more these two vectors i and j, in the moment we find another component of i that dominates its corresponding of j.

4.4 Results and Comments

The results presented in this section are separated into two categories. The first collection is the output of the first algorithm, while the second one is the output of the improved algorithm.

In the very beginning the algorithm was tested with small problems whose solutions were already known by solving them at the same time by a complete enumeration scheme. The data was entered in a file that was called by the program.

Once the algorithm was working properly, all the problems were generated randomly in the interval 0-100, except the RHS of the constraint set that was fixed to a constant K times the sum of the coefficients of the row, (K was 0.25, 0.50 and 0.75).

The time consumed in each problem was found by computing the difference between the CPU time before starting the subroutine "Algorithm" and the CPU time just when the algorithm was finished and the subroutine returned to the main program to generate another problem.

4.4.1 Results of the First Algorithm

A first set of results is given in the tables #1, #2 and #3. The problems were generated randomly with all components of C and A (objectives and constraints) positives.

The size of the problem was: P, the number of objective functions equal 3; M, the number of constraints equal 8; and N, the number of variables equal to 10, 14, 18 and 20.

Five problems were generated in each case, for different

TABLE #1: RESULTS

P=3	K=0.75	T	NPA	NEP
N=10		3.96	102	10
M=8		35.26	92	25
		6.84	71	4
		10.08	62	6
		24.48	107	17
	K=0.50	T	NPA	NEP
		23.76	90	16
		15.48	54	10
		11.16	59	7
		11.16	50	7
		15.48	79	10
	K=0.25	T	NPA	NEP
		5.76	13	3
		8.64	13	5
		10.08	9	6
		8.64	13	5
		2.88	9	1

TABLE #2: RESULTS

P=3	K=0.75	T	NPA	NEP
N=14		73.44	378	46
M=8		16.92	168	11
		37.44	233	26
		24.48	222	15
		32.40	240	19
	K=0.50	T	NPA	NEP
		42.12	541	22
		36.36	319	24
		22.32	236	15
		42.12	430	25
		90.0	507	26
	K=0.25	T	NPA	NEP
		11.16	61	7
		9.00	59	5
		7.20	33	4
		6.21	61	3
		17.28	72	12

TABLE #3: RESULTS

P=3	K=0.25	T	NPA	NEP
N=18		26.64	228	23
M=8		25.92	348	20
		28.80	312	21
		38.16	265	28
		24.48	332	18

P=3	K=0.25	T	NPA	NEP
N=20		32.40	412	14
M=8		71.64	640	16
		54.72	523	28
		29.16	382	12
		74.16	745	36

AVERAGE	K=0.25	T	NPA	NEP
N=10		7.2	11	3
N=14		10.15	57	5
N=18		28.80	297	22
N=20		52.42	540	21

Values of the RHS. The RHS was a factor K times the sum of the coefficient of the respective row. We vary K from 0.25, 0.50 and 0.75.

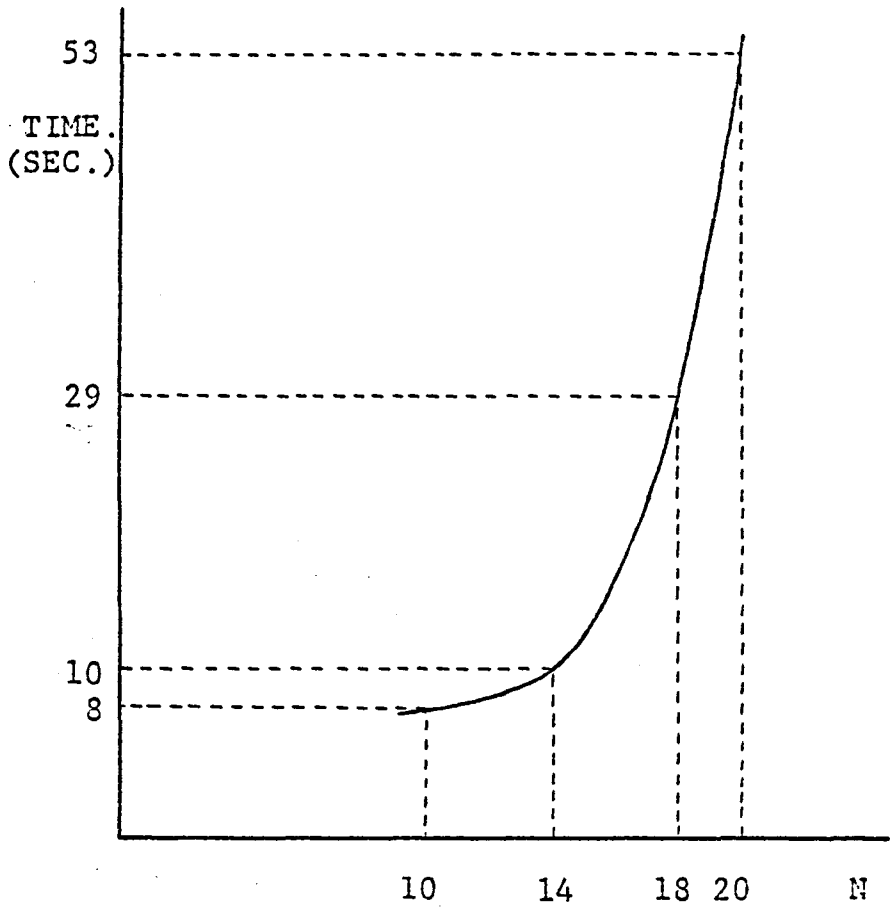
The principal problem with this algorithm is its number of feasible points (five to ten times the number of efficient points) which give us problems of computer capacity.

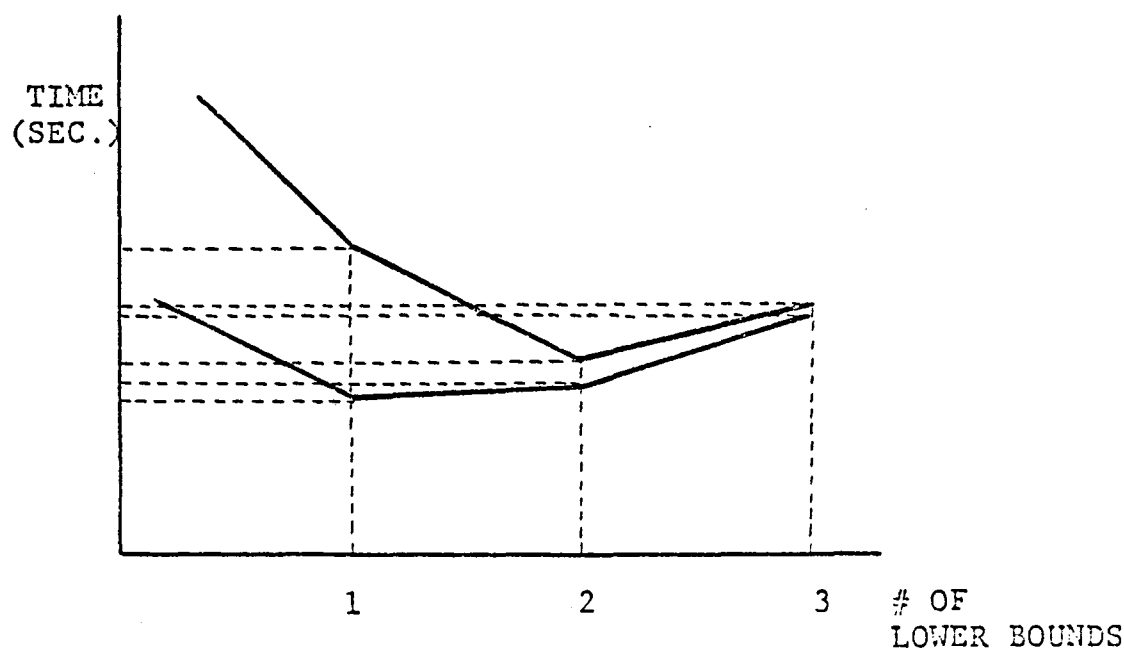
T is the solution time in seconds, NPA is the number of points analyzed (feasible included in the list) and NEP is the total number of efficient points of the problem.

The problems done in the least amount of CPU time were undoubtedly for K=0.25 whats seems reasonable given our method of branching. K=0.75 problems gave worse computational results but we could reduce their CPU time by branching in just the opposite way that we did. K=0.50 problems were the worst of all cases analyzed. Given that result, we continued testing problems with K=0.25, increasing the number of variables.

In Table #3, we analyze problems with RHS equal to 0.25 times the sum of the coefficients of the rows. The number of

FIGURE 17. VARIABLES VS. SOLUTION TIME





variables was increased to 18 and 20.

The last graph shows how the number of variables increases the solution time. Therefore, our next goal is to try to decrease this CPU time, either by improving the algorithm or changing the structure of the problem. (Figure 17).

4.4.2 Results of the Improved Algorithm

The results of the improved algorithm are presented in Tables #4 through #10.

Two questions have to be answered at this point. How many lower bounds do we need to find by the λ's method, and where do we have to begin the bounding procedure?

In Table #4, we test different problems for one and for two lower bounds. The inverse dominance test is performed each time we find a feasible point. The result is invariably an increase in the solution time. In some cases, generally in problems with K=0.25, we obtained the same lower bound for different λ's, in part due to the small number of possible

TABLE #4 : RESULTS

P=3	K=0.75	ONE LB	TWO LB
N=14		66.60	67.68
M=4		25.20	28.44
		33.12	41.04
		29.88	33.48
		48.60	53.04
	K=0.50	ONE LB	TWO LB
		83.52	126.00
		18.72	42.48
		22.68	43.20
		35.64	56.88
		136.44	163.44
	K=0.25	15.12	19.44
		13.32	15.84
		16.20	19.08
		11.88	14.76
		31.68	34.92

efficient points of this case.

We believe that the larger the number of efficient points of a problem, the larger the number of lower bounds that we can find without increasing the solution time.

One positive fact was that the lower bound found was not only efficient but a good efficient, in the sense that it dominates a large number of feasible points.

In Table #5 we solve a series of problems, changing the beginning of the bounding procedure at different levels of the algorithm as a function of N, the number of variables of the problem. The inverse dominance test is performed each time we find a feasible point and only one lower bound is found by the 's method.

We begin this test at levels 0, N/2, N/4, and N-5. The number of times bounded was the same in all the cases, which tells us that the bounding test was not really bounding although the reduction in CPU time was in some cases of more than half the time employed in solving the problem

TABLE #5: RESULTS

P=3 K=0.25

N=20

M=8

BOUNDING AT THE BEGINNING

T	NPA	NEP
27.00	49	14
42.48	52	21
29.16	39	12
30.60	32	14
63.72	82	36

BOUNDING AFTER N/2

T	NPA	NEP
26.28	49	14
41.76	52	21
27.72	39	12
29.88	32	14
62.64	82	36

BOUNDING AFTER N-5

T	NPA	NEP
16.72	49	14
19.00	52	21
16.76	50	12
14.80	32	14
21.30	82	36

Over the N-5 level, the number of times bounded went up due to the effect of bounding late, given that the number of branches increases by a factor of two in each step. For example, if one branch were to be bounded at stage k of the problem but instead we bound at stage k+1, we will have to bound twice instead of once. At stage k+2 we will have to bound four times and so on.

As we have discussed, the problem of bounding tests is the upper bound. Given that the lower bound found by the λ's method was very good, the only explanation for this failure is that the upper bound was very large. A way to solve this, is by finding a tighter upper bound by any of the other methods dicussed in Chapter III, taking always into account the problem of the time consumed in finding these new upper bounds.

In Tables #6-10 we analyze the performances of the second algorithm. The improvements over the first algorithm are again:

-inverse dominance each time

-1 LB found by the λ's method

-bounding after the (N-5) variable

TABLE #6: RESULTS

P=3	K=0.75	T	NPA	NEP
N=10		2.02	43	10
M=8		2.09	36	25
		1.35	22	4
		1.47	17	6
		2.23	40	18
	K=0.50	T	NPA	NEP
		1.50	35	14
		1.06	10	10
		0.93	20	7
		0.98	10	7
		1.32	31	10
	K=0.25	T	NPA	NEP
		0.22	8	3
		0.25	8	5
		0.22	6	6
		0.22	8	5
		0.17	2	1

TABLE #7 : RESULTS

P=3	K=0.75	T	NPA	NEP
N=14		20.08	111	55
M=8		13.80	30	11
		15.63	61	26
		15.36	44	15
		17.08	62	20
	K=0.50	T	NPA	NEP
		15.26	118	32
		10.82	45	20
		8.94	44	15
		14.70	87	22
		17.34	109	62
	K=0.25	T	NPA	NEP
		1.03	15	7
		1.08	7	5
		0.73	7	4
		1.09	12	3
		1.25	20	11

TABLE #8: RESULTS

P=3	K=0.75	T	NPA	NEP
N=18		355	-	26
M=8		223	-	36
		194	-	28
		204	-	62
		246	-	44
	K=0.50	T	NPA	NEP
		216	-	22
		195	228	54
		148	179	31
		138	119	51
		195	232	36
	K=0.25	T	NPA	NEP
		6.39	54	23
		8.20	37	13
		8.00	90	41
		8.00	40	18
		8.00	51	20

TABLE #9: RESULTS

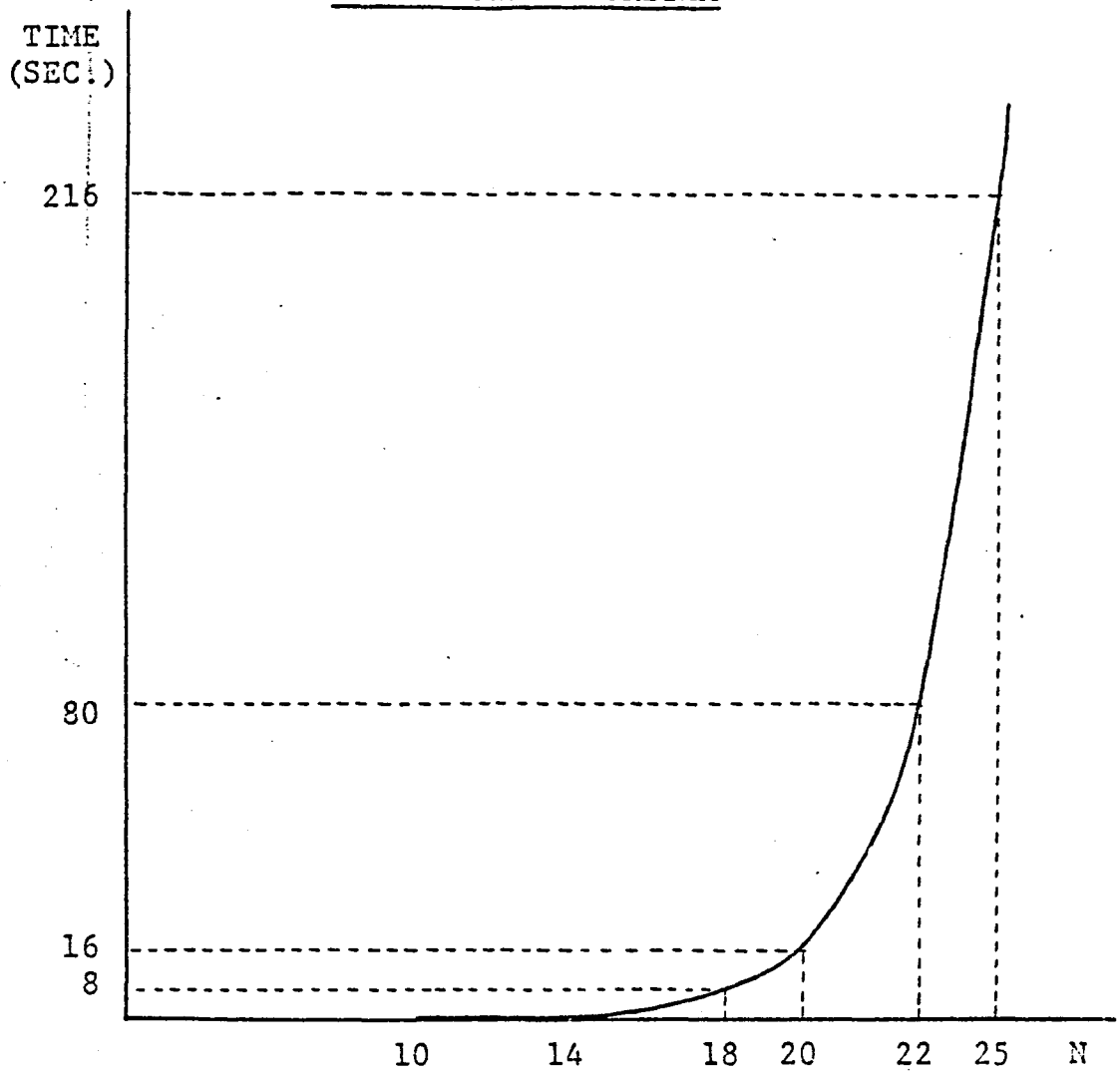
P=3	K=0.25	T	NPA	NEP
N=20		16.72	49	14
M=8		19.00	52	21
		16.76	50	12
		14.80	32	14
		21.30	82	36

P=3	K=0.25	T	NPA	NEP
N=23		66.70	81	24
M=8		72.40	67	16
		80.00	117	50
		77.30	54	23
		104.40	171	60

P=3	K=0.25	T	NPA	NEP
N=25		229	-	50
M=8		215	-	45
		170	-	28
		250	-	56
		290	-	76

FIGURE 18.

THE SECOND ALGORITHM



	TIME (SEC.)
N=10	0.22
N=14	1.04
N=18	7.72
N=20	17.20
N=23	80.16
N=25	216.22

With this improved algorithm, we have reduced the number of feasible points substantially, to a level of an average of two or three times the number of efficient points which reduces our problem of computer capacity.

In table #9 and #10, we analyze the problem for K=0.25, increasing the number of variables to 20, 23 and 25. It seems clear, after seeing in Figure 18 that the solution time is increased exponentially with the number of variables. What is not so clear, is how the number of constraints will increase the solution time. Will it be linear as in the case of only one objective, or will it also be exponential?

Another interesting issue to analyze is the number of variables set at one in a typical efficient point. Below, we give a list of the number of one's versus the size of the problem:

Since our branching rule preferentially sets variables to 0, our algorithm works better for 0.25 problems. If we change the branching rule to 1, the algorithm will work better for 0.75 problems.

P=3	APPROXIMATE NUMBER OF VARIABLES SET AT 1		
M=8	K=0.75	K=0.50	K=0.25
N=10	6	3	1
N=14	9	5	2
N=18	10	7	3
N=20	-	-	4
N=23	-	-	5
N=25	-	-	6

4.5 Comparisons and Suggestions

No one at the present moment has been able to solve problems up to 25 variables and 8 constraints. Bitran [10] has solved problems of the size P=3, N=18 and M=4, and Villarreal and Karwan [44] problems with P=3, N=14 and M=4. We conclude then, that the algorithm has been a relative success, although the solution time at N=25 was already very large. The real drawback of the algorithm is for K=0.50 of the sum of the coefficients of the rows. In this case we only could arrive at 18 variables.

We have also seen that the algorithm increases the solution time exponentially with the number of variables, and most likely also with the number of constraints. The future development of this algorithm has to shift this point as far as possible. Avenues for future research are the determination of better upper bounds of the free variables and the analysis of problems with special structures.

Another topic to be explored and that could be rewarding is a decomposition technique that could permit the solution of problems through a sequence of smaller problems, taking advantage of the significant reduction in computing time.

REFERENCES

1. Agarwal, S.K., "Optimizing Techniques for the Interactive Design of Transportation Networks under Multiple Objectives" Ph. D. Dissertation, Northwestern University, 1973.

2. Agin, N., "Optimum Seeking with Branch-and-Bound," Management Science 13, (1966), B.176-187.

3. Balas, E., "An Additive Algorithm for Solving Linear Programs With Zero-One Variables," Operations Research 13, (1965), 517-546.

4. Balas, E., "Discrete Programming by the Filter Method," Operations Research 15, (1967), 915-957.

5. Balinski, M.L., "Integer Programming: Methods, Uses, Computation," Management Science 12, (1965), 253-313.

6. Beale, E.M.L., and Small, R., "Mixed Integer Programming by a Branch-and-Bound Technique," Procedings of the IFIP Congress, Volume 2. (1965), 450-451.

7. Bell, D.E. and Shapiro, J.F., "A Finitely Convergent Duality Theory for Zero-One Integer Programming," MIT Operations Research Center Report OR 043-75, (1975).

8. Benayoun, R., Montgolfier, J., Tergny, J. and Laritcher, O. "Linear Programming with Multiple Objectives: Step Method" Mathematical Programming, Vol. 1, (1971).

9. Bitran, G.R., "Linear Multiple Objective Programs with Zero-One Variables," Mathematical Programming 13, (1977), 121-139.

10. Bitran, G.R., "Theory and Algorithms for Linear Multiple Objective Programs with Zero-One Variables," Technical Report No. 150, Operations Research Center, MIT, May 1978.

11. Breu, R., and Burdet, C., "Branch and Bound Experiments in Zero-One Programming," Mathematical Programming Study 2, (1974).

12. Dakin, R., "A Tree-Search Algorithm for Mixed Integer Programming Problems," Computer Journal 8, (1965), 250-255.

13. Davis, R., "Branch-and-Bound Algorithm for Zero-One Mixed Integer Programming Problems," Operations Research 19, (1971), 1036.

14. Driebeek, N.J., "An Algorithm for the Solution of Mixed Integer Programming Problems," Management Science 12, (1966), 576.

15. Evans, J.P., and Steuer, R.E., "A Revised Simplex Method For Linear Multiple Objective Programs," Mathematical Programming 5, (1973), 54-72.

16. Fleischmann, B., "Computational Experience with the Algorithm of Balas," Operations Research 15, (1967), 153-155.

17. Freeman, R.J., "Computational Experience with the Balas Integer Programming Algorithm," paper P-3241, The RAND Corporation, October 1965.

18. Garfinkel, R.S., and Nemhauser, G.L., Integer Programming (Wiley, New York. 1972).

19. Geoffrion, A.M., "Lagrangean Relaxation for Integer Programming," Mathematical Programming study 2, (1974).

20. Geoffrion, A.M., "Integer Programming by Implicit Enumeration and Balas' Method," SIAM Review 9, (1967), 178-190.

21. Geoffrion, A.M., "An Improved Implicit Enumeration Approach for Integer Programming," Operations Research 17, (1969), 437-454.

22. Geoffrion, A.M., "Proper Efficiency and The Theory of Vector Maximization," Journal of Mathematical Analysis and Applications 22, (1968), 618-630.

23. Geoffrion, A.M., and Marsten, R.E., "Integer Programming Algorithms: A Framework and State-of-the-art Survey," Management Science 18, (March, 1972).

24. Glover, F., and Zionts, S., "A Note on the Additive Algorithm of Balas," Operations Research 13, (1965), 546-549.

25. Glover, F., "Surrogate Constraints," Operations Research 16, (1968), 741-749.

26. Glover, F., "Surrogate Constraint Duality in Mathematical Programming," Operations Research 23, (1975), 435.

27. Gomory, R.E., "An Algorithm for the Mixed Integer Problem," The RAND Corporation, RM-2597. (1960).

28. Greenberg, H.J., "The Generalized Penalty Function Surrogate Model," Operations Research 21, (1973), 162-178.

29. Greenberg, H.J., and Pierskalla, W.P., "Surrogate Mathematical Programs," Operations Research 18, (1970), 924-939.

30. Isermann, H., "Proper Efficiency and The Linear Vector Maximum Problem," Operations Research 22, (1974),.

31. Kendall, K., and Zionts, S., "Solving Integer Programming Problems By Aggregation Constraints," Operations Research 25, (1977), 346.

32. Kornbluth, J.S.H., "Duality, Indifference and Sensitivity Analysis in Multiple Objective Programming," Operations Research Quarterly 25, (1974), 599-614.

33. Land, H.A., and Doig, A.G., "An Automatic Method of Solving Discrete Programming Problems," Econometrica 28, (July 1960), 497-520.

34. Lawler, E.L., and Wood, D.E., "Branch-and-Bounds Methods: A Survey," Operations Research 14, (1966), 699-719.

35. Lemke, C.E., and Spielberg, K., "Direct Search Algorithms for Zero-One and Mixed Integer Programming," Operations Research 15, (1967), 892.

36. Loulou, R., and Michaelides, E., "New Greedy-Like Hueristics for the Multidimensional Zero-One Knapsack Problem," Working Paper, McGill University, (1977).

37. Marsten, R.E., and Morin, T.L., "A Hybrid Approach to Discrete Mathematical Programming," Mathematical Programming 14, (1978).

38. Mitten, L., "Branch-and-Bounds Methods: General Formulation and Properties," Operations Research 18, (1970), 24.

39. Pasternak, H., and Passy, U., "Bicriterion Mathematical Programs with Boolean Variables," in J. Cochrane and M. Zeleny (eds). Multiple Criteria Decision Making, U. of South Carolina Press, (1973), 327-348.

40. Philip, J., "Algorithms for the Vector Maximization Problem," Mathematical Programming 2, (1972), 207-229.

41. Shapiro, J.F., "Multiple Criteria Public Investment Decision Making By Mixed Integer Programming," in Proc. of The Conference On Multicriterion Decision MakingJouy En Josas, May 1975, to appear.

42. Tomlin,J., "An Improved Branch-and-Bound Method for Integer Programming," Operations Research 19, (1971), 1070.

43. Toyoda, Y., "A Simplified Algorithm for Obtaining Approximate Solutions to Zero-One Programming Problems," Management Science 21, (1975).

44. Villarreal, B., and Karwan, M.H., "Dynamic Programming Approaches for Multicriteria Integer Programming," Research Report No. 78 - 3, The Operations Research Program, Department of Industrial Engineering, State University of New York in Buffalo, New York, (1977).

45. Yu, P.L., and Zeleny, M., "The Set of All Nondominated Solutions in Linear Cases and in Multicriteria Simplex Method," Journal of Mathematical Analysis and Applications 49, (1975), 430-468.

46. Zeleny, M., "Compromise Programming," in J. Cochrane and M. Zeleny (eds) Multiple Criteria Decision Making, U. of South Carolina Press, (1973), 262-301.

47. Zionts, S., "Integer Linear Programming with Multiple Objectives," Annals of Discrete Mathematics 1, (1977).

48. Zionts, S., "Generalized Implicit Enumeration Using Bounds on Variables for Solving Linear Programs with Zero-One Variables," Naval Res. Log. Quarterly 19, (1972), 165.

49. Zionts, S., and Wallenius, J., "Identifying Efficient Vectors: Some Theory and Computational Results," Working Paper No. 257, State University of New York at Buffalo School of Management, New York, (1976).

APPENDIX

ALGORITHM PRINTOUT AND TYPICAL OUTPUT

In pages 165 through 170 we present the printouts for the algorithm described in section 4. The algorithm is presented as a subroutine of a Main Program (page 171) which is the generator of random problems.

The resulting output is shown on page 172. The imputs are the following: P, the number of objective functions, M, the number of constraints, and N, the number of variables. F is the beginning of the interval in which the random problem is generated and D is the constant that, multiplied by the sum of the coefficient of each row, gives us the RHS.

In the typical output shown on page 172, the lower bound indicates the first lower bound found. Besides the efficient points and the total time, the figure 49 indicates the total number of feasible points included in the list before testing direct dominance.

```
      SUBROUTINE PEPE
      COMMON P,N,M,C,A,B
      INTEGER H,P,R,Q,V,VAR,VAL,F,S
       DIMENSION C(30,30),A(30,30),V(30,30),EF(200,30),B(30),
     1SA(30),LA(5,5),SC(30),S(30),SAN(30),SAP(30),VAR(30),VAL(30),
     2JP(30),JQ(30),UBF(30),UB(30),Z(30)
501   FORMAT(3I10,5F10.2/50(8F10.2/))
      LA(1,1)=1
      LA(2,1)=1
      LA(3,1)=1
      Q=1
      DO 13 R=1,Q
      DO 14 J=1,N
14    V(R,J)=0
13    CONTINUE
      DO 19 R=1,Q
      DO 10 J=1,N
      SC(J)=0.
      DO 11 H=1,P
11    SC(J)=SC(J)+LA(H,R)*C(H,J)
10    CONTINUE
   15 CM=0.
      DO 12 J=1,N
      IF(V(R,J).EQ.1) GO TO 12
      IF(CM.GE.SC(J)) GO TO 12
      CM=SC(J)
      MJC=J
12    CONTINUE
      J=MJC
      V(R,J)=1
C     LOOKING FOR FEASIBILITY
      DO 16 I=1,M
      SA(I)=0.
      DO 17 J=1,N
      IF (V(R,J).EQ.0) GO TO 17
      SA(I)=SA(I)+A(I,J)
   17 CONTINUE
      IF (SA(I).GT.B(I)) GO TO 18
   16 CONTINUE
      GO TO 15
18    J=MJC
      V(R,J)=0
      WRITE(1,7)
7     FORMAT('LOWER BOUND')
      WRITE(1,511)(V(R,J),J=1,N)
19    CONTINUE
C     FINDING A SET OF LOWER BOUNDS
      DO 20 R=1,Q
      DO 21 H=1,P
      EF(R,H)=0.
      DO 22 J=1,N
      IF (V(R,J).EQ.0) GO TO 22
      EF(R,H)=EF(R,H)+C(H,J)
   22 CONTINUE
   21 CONTINUE
   20 CONTINUE
```

```
511   FORMAT(10(1H ,I5),/)
711   FORMAT(10(1H ,F10.2),/)
C     INITIALIZATION
502   FORMAT(3I10)
29    LEV=0
      DO 23 J=1,N
   23 S(J)=0
      DO 24 H=1,P
   24 Z(H)=0.
C     PERFORMING FEASIBILITY
55    IS=0
      DO 58 I=1,M
      IF (B(I).GE.0.) GO TO 43
      IS=1
C     FINDING ISAN
      SAN(I)=0.
      DO 27 J=1,N
      IF (S(J).EQ.1) GO TO 27
      IF (A(I,J).GE.0.) GO TO 27
      SAN(I)=SAN(I)+A(I,J)
27    CONTINUE
      IF(B(I)-SAN(I))44,45,46
   44 F=0
      GO TO 60
   45 DO 50 J=1,N
      IX=I
      IF (S(J).EQ.1) GO TO 50
      IF (A(I,J).EQ.0.) GO TO 50
      LEV=LEV+1
      S(J)=1
      IF (A(I,J).GT.0.) GO TO 49
      VAR(LEV)=J
      VAL(LEV)=3
C     CALL DATE1(J)
      DO 101 H=1,P
  101 Z(H)=Z(H)+C(H,J)
      DO 104 I=1,M
  104 B(I)=B(I)-A(I,J)
      I=IX
      GO TO 50
   49 VAR(LEV)=J
      VAL(LEV)=2
   50 CONTINUE
      GO TO 55
C     PERFORMING IHQ
   46 AQ=0.
      DO 30 J=1,N
      IF (S(J).EQ.1) GO TO 30
      IF (AQ.LE.A(I,J)) GO TO 30
      AQ=A(I,J)
      JQ(I)=J
   30 CONTINUE
```

```
      IF(AQ.EQ.0.) GO TO 57
      C1=B(I)-SAN(I)+AQ
      XH=C1/AQ
      IF (XH.LE.0.) GO TO 57
      IF (XH.GT.1.) GO TO 44
      LEV=LEV+1
      J=JQ(I)
      S(J)=1
      VAR(LEV)=J
      VAL(LEV)=3
C     CALL DATE1(J)
      DO 102 H=1,P
  102 Z(H)=Z(H)+C(H,J)
      DO 105 I=1,M
  105 B(I)=B(I)-A(I,J)
      GO TO 55
C     PERFORMING ISAMP
   43 SAN(I)=0.
      SAP(I)=0.
      DO 35 J=1,N
      IF (S(J).EQ.1) GO TO 35
      IF (A(I,J)) 33,35,34
   33 SAN(I)=SAN(I)+A(I,J)
      GO TO 35
   34 SAP(I)=SAP(I)+A(I,J)
   35 CONTINUE
      IF(SAP(I).EQ.0.) GO TO 58
C     PERFORMINF IUP
   57 AP=0.
      DO 36 J=1,N
      IF (S(J).EQ.1) GO TO 36
      IF (AP.GE.A(I,J)) GO TO 36
      AP=A(I,J)
      JP(I)=J
   36 CONTINUE
      IF(AP.EQ.0.) GO TO 58
      C2=B(I)-SAN(I)
      XU=C2/AP
      IF (XU.GE.1.0) GO TO 58
      IF (XU.LT.0.0) GO TO 44
      J=JP(I)
      LEV=LEV+1
      VAR(LEV)=J
      VAL(LEV)=2
      S(J)=1
      GO TO 55
   58 CONTINUE
      F=1
   60 IF(F.EQ.1) GO TO 100
  250 IF (LEV.EQ.0) GO TO 200
```

```
C       BACKTRACKING
  150   J=VAR(LEV)
        IF (VAL(LEV).EQ.0) GO TO 61
        IF (VAL(LEV).EQ.1) GO TO 62
        S(J)=0
        IF (VAL(LEV).EQ.2) GO TO 63
C       CALL DATE2(VAR(LEV))
        DO 111 H=1,P
  111   Z(H)=Z(H)-C(H,J)
        DO 113 I=1,M
  113   B(I)=B(I)+A(I,J)
   63   LEV=LEV-1
        IF(LEV) 200,200,150
   61   VAL(LEV)=3
C       CALL DATE1(J)
        DO 103 H=1,P
  103   Z(H)=Z(H)+C(H,J)
        DO 106 I=1,M
  106   B(I)=B(I)-A(I,J)
        GO TO 65
   62   VAL(LEV)=2
C       CALL DATE2(VAR(LEV))
        DO 112 H=1,P
  112   Z(H)=Z(H)-C(H,J)
        DO 114 I=1,M
  114   B(I)=B(I)+A(I,J)
   65   GO TO 55
  100   IF(LEV.EQ.N) GO TO 300
        IF(LEV.LE.(N-5)) GO TO 59
C       BOUNDING
        IF(Q.EQ.0) GO TO 59
        DO 68 H=1,P
        UBF(H)=0.
        DO 69 J=1,N
        IF (S(J).EQ.1) GO TO 69
        IF (C(H,J).LE.0.) GO TO 69
        UBF(H)=UBF(H)+C(H,J)
   69   CONTINUE
        UB(H)=Z(H)+UBF(H)
   68   CONTINUE
        DO 71 R=1,Q
        DO 72 H=1,P
        IF(UB(H).GT.EF(R,H)) GO TO 71
   72   CONTINUE
        IB=1
        GO TO 73
   71   CONTINUE
        IB=0
   73   IF (IB.EQ.1) GO TO 250
```

```
C       BRANCHING
   59   IF (IS.EQ.0) GO TO 81
        I=1
   77   IF (B(I).LT.0.) GO TO 76
        I=I+1
        GO TO 77
   76   XM1=B(I)-SAN(I)
        IM=I
   78   I=I+1
        IF (I.GT.M) GO TO 80
        IF (B(I).GE.0.) GO TO 78
        IF (XM1.LE.(B(I)-SAN(I))) GO TO 78
        XM1=B(I)-SAN(I)
        IM=I
        GO TO 78
   80   I=IM
        J=JQ(I)
        LEV=LEV+1
        VAR(LEV)=J
        VAL(LEV)=0
        S(J)=1
        GO TO 90
   81   I=1
   83   IF (B(I).GE.0.) GO TO 82
        I=I+1
        GO TO 83
   82   XM2= B(I)-SAN(I)-SAP(I)
        IM=I
   85   I=I+1
        IF (I.GT.M) GO TO 86
        IF (B(I).LT.0.) GO TO 85
        IF (XM2.LE.(B(I)-SAN(I)-SAP(I))) GO TO 85
        XM2=B(I)-SAN(I)-SAP(I)
        IM=I
        GO TO 85
   86   I=IM
        IF(SAP(I).NE.0.) GO TO 88
        DO 89 J=1,N
        IF(S(J).EQ.0)GO TO 87
89      CONTINUE
   88   J=JP(I)
   87   LEV=LEV+1
        VAR(LEV)=J
        VAL(LEV)=1
        S(J)=1
C       CALL DATE1(J)
        DO 201 H=1,P
  201   Z(H)=Z(H)+C(H,J)
        DO 202 I=1,M
  202   B(I)=B(I)-A(I,J)
   90   GO TO 55
```

```
C      PERFORMING LB(EF)
  300  Q=Q+1
       R=Q
       DO 91 H=1,P
   91  EF(R,H)=Z(H)
       IF(Q.EQ.1) GO TO 150
       NQ=Q-1
       DO 166 R=1,NQ
       DO 167 H=1,P
       IF(EF(R,H).LT.EF(Q,H)) GO TO 166
167    CONTINUE
       Q=Q-1
       GO TO 150
166    CONTINUE
       GO TO 150
C      PERFORMING EFFI
200    IF(Q.NE.0) GO TO 225
       WRITE(1,226)
226    FORMAT('THERE NO EFFICIENTS')
       GO TO 227
225    IF(Q.EQ.1) GO TO 517
       WRITE(1,933)
933    FORMAT('         Q')
       WRITE(1,502)Q
       R=1
92     L=R+1
  140  H=1
       IL=0
94     IF(EF(R,H).GT.EF(L,H)) GO TO 97
       IF(EF(R,H).EQ.EF(L,H))GO TO 155
       IL=1
155    H=H+1
       IF (H.LE.P) GO TO 94
       DO 96 H=1,P
   96  EF(R,H)=EF(Q,H)
       Q=Q-1
       GO TO 152
97     IF(IL.EQ.1) GO TO 93
   99  IF (EF(R,H).LT.EF(L,H)) GO TO 93
       H=H+1
       IF (H.LE.P) GO TO 99
       DO 98 H=1,P
   98  EF(L,H)=EF(Q,H)
       Q=Q-1
       GO TO 153
   93  L=L+1
  153  IF(L.LE.Q) GO TO 140
       R=R+1
  152  IF(R.LE.(Q-1)) GO TO 92
       WRITE(1,517)
517    FORMAT('EFFICIENTS')
       DO 524 R=1,Q
524    WRITE(1,711)(EF(R,H),H=1,P)
227    RETURN
       END
```

MAIN PROGRAM

```
      INTEGER P,PP
      COMMON P,N,M,C,A,B
      DIMENSION C(30,30),A(30,30),NN(30),FF(30),SA(30),B(30),DK(30)
      NN(1)=20
      NN(2)=23
      NN(3)=25
      FF(1)=.1
      FF(2)=.3
      FF(3)=.5
      FF(4)=.7
      FF(5)=.9
      P=3
      M=8
      D=.25
      DO 10 L=1,2
      N=NN(L)
      DO 12 K=1,1
      F=FF(K)
      WRITE(1,1)P,N,M,F,D
      X=RAND$A(F)
      DO 14 I=1,P
      DO 15 J=1,N
      X=RAND$A(X)
      X=100*X
15    C(I,J)=X
14    CONTINUE
      DO 16 I=1,M
      SA(I)=0.
      DO 17 J=1,N
      X=RAND$A(X)
      X=100*X
      A(I,J)=X
      SA(I)=SA(I)+A(I,J)
17    CONTINUE
      B(I)=D*SA(I)
16    CONTINUE
      WRITE(1,4)
4     FORMAT('          P          N          M          F          D')
1     FORMAT(3I10,2F10.2)
      T1=CTIM$A(ICP)
2     FORMAT(F10.2)
      CALL PEPE
      T2=CTIM$A(ICP)
      T=T2-T1
      WRITE(1,50)
50    FORMAT('TOTAL TIME')
      WRITE(1,2)T
12    CONTINUE
10    CONTINUE
      STOP
      END
```

TYPICAL OUTPUT

```
  TYPE OUTPUT

C>R *PACD
          3          20          8        0.10        0.25
          P           N          M           F           D
1LOWER BOUND
     0    0     0     0     0     1     0     0     0     0

     0    0     0     1     0     0     0     0     0     0

          Q
         49
 EFFICIENTS
      232.97      311.51      309.04
      208.05      309.46      318.55
      170.32      334.60      306.03
      244.27      300.94      224.43
      348.55      156.38      249.40
      272.50      285.03      309.91
      265.46      299.38      267.62
      345.47      167.29      266.52
      203.28      336.03      298.99
      214.07      265.01      331.01
      302.95      249.06      339.40
      295.91      263.41      297.11
      333.34      244.52      346.88
      236.39      319.34      304.44
      312.16      246.08      303.68
 TOTAL TIME
      17.16
```

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-159365		

4. Title and Subtitle	5. Report Date
Theoretical Study of Network Design Methodologies for the Aerial Relay System	June 1980
	6. Performing Organization Code: 3140
7. Author(s): Jorge M. Rivera and Robert W. Simpson	8. Performing Organization Report No.: FTL R80-10
9. Performing Organization Name and Address: Flight Transportation Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139	10. Work Unit No.
	11. Contract or Grant No.: NAS1-15268
12. Sponsoring Agency Name and Address: National Aeronautics and Space Administration, Langley Research Center, Hampton, VA 23665	13. Type of Report and Period Covered: Contractor Report
	14. Sponsoring Agency Code: 530-04-13

15. Supplementary Notes
Technical Representative: A.C. Kyser, ASD
NASA Langley Research Center
Hampton, VA 23665

16. Abstract

The Aerial Relay System network design problem is discussed. A generalized branch and bound based algorithm is developed which can consider a variety of optimization criteria, such as minimum passenger travel time and minimum liner and feeder operating costs. The algorithm, although efficient, is basically useful for small-size networks, due to its nature of exponentially increasing computation time with the number of variables.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Aerial Relay System Operations Research Branch and Bound Algorithm	Unclassified, Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
unclassified	unclassified	172	

For sale by the National Technical Information Service, Springfield, Virginia 22161

End of Document